K-means Clustering Report

Author: Rodrigo Santos de Carvalho - Published 2021-12-06 15:08 - (0 Reads)

K-means clustering in Machine Learning

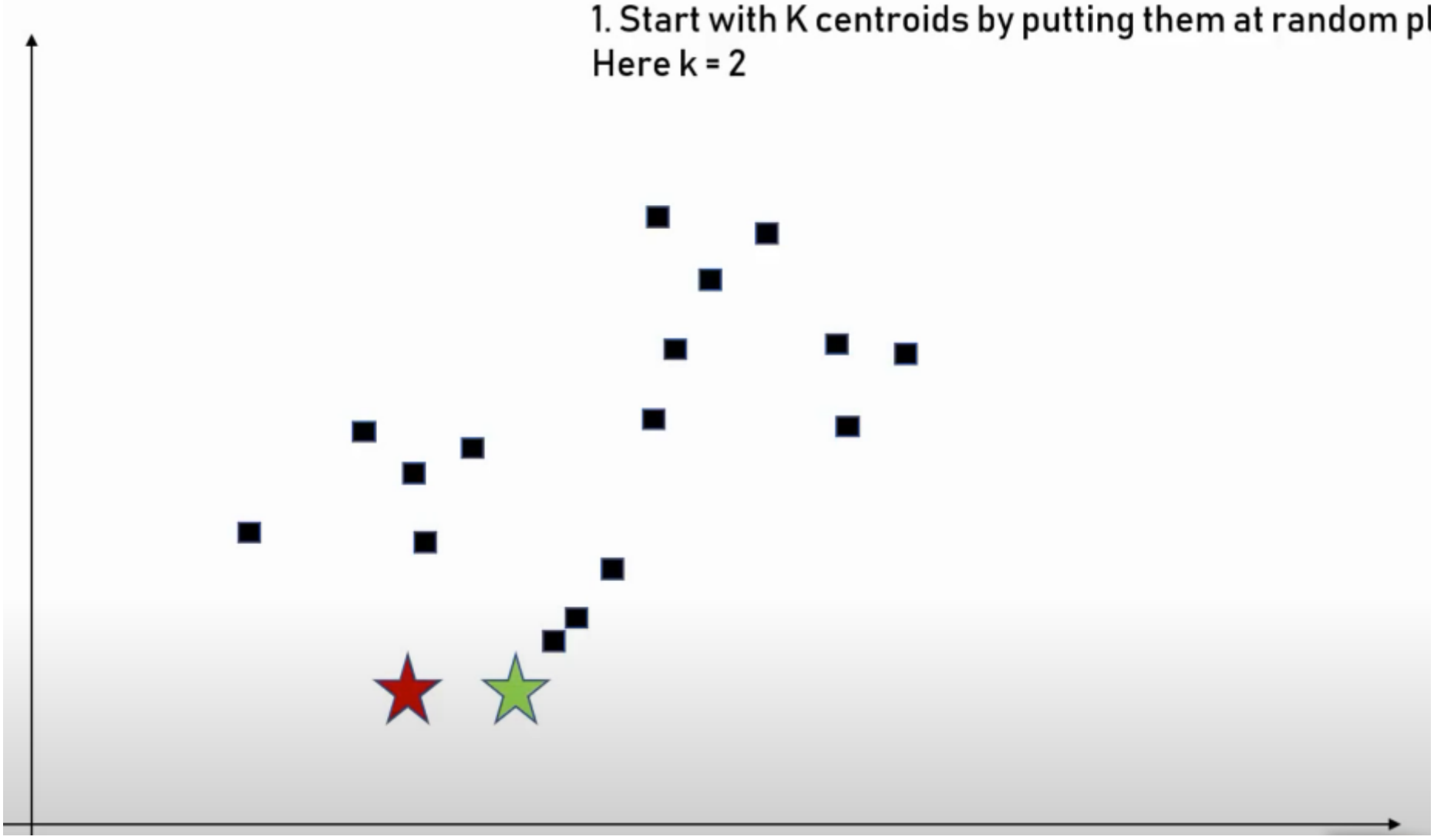
Given the large amount of data that's usually involved in Machine Learning, to have a better understanding of it, clustering is used often. Clustering basically consists in grouping unlabeled data into clusters (groups); it can be extremely helpful in classifying data into structures that are more easily understood and manipulated. Therefore, to perform clustering, K-Means clustering is one of the simplest and popular unsupervised machine learning algorithms. This report covers the K-Means algorithm and a course example in python.

K-means Clustering Algorithm

As a clustering algorithm, K-means' objective is grouping data points, which is done based on the distance between a data point and the cluster centroids (the beginning points for every cluster). The cluster with the closest centroid to a data point is the one to which that data point will belong. It starts with a group of randomly selected centroids and repeats the calculations to find better positions for the centroids; said calculations consider the means of the data points' coordinates . The algorithm stops when the clustering is successful (the centroids have stabilized) or when the established number of repetitions has been achieved.

To understand that algorithm, it'll be applied to the dataset below.

First, the free parameter "k" must be defined, for it is the number of clusters in the dataset. In this example, "k" is defined as 2. And the first group of centroids are randomly created. In the image below, the stars represent the centroids.

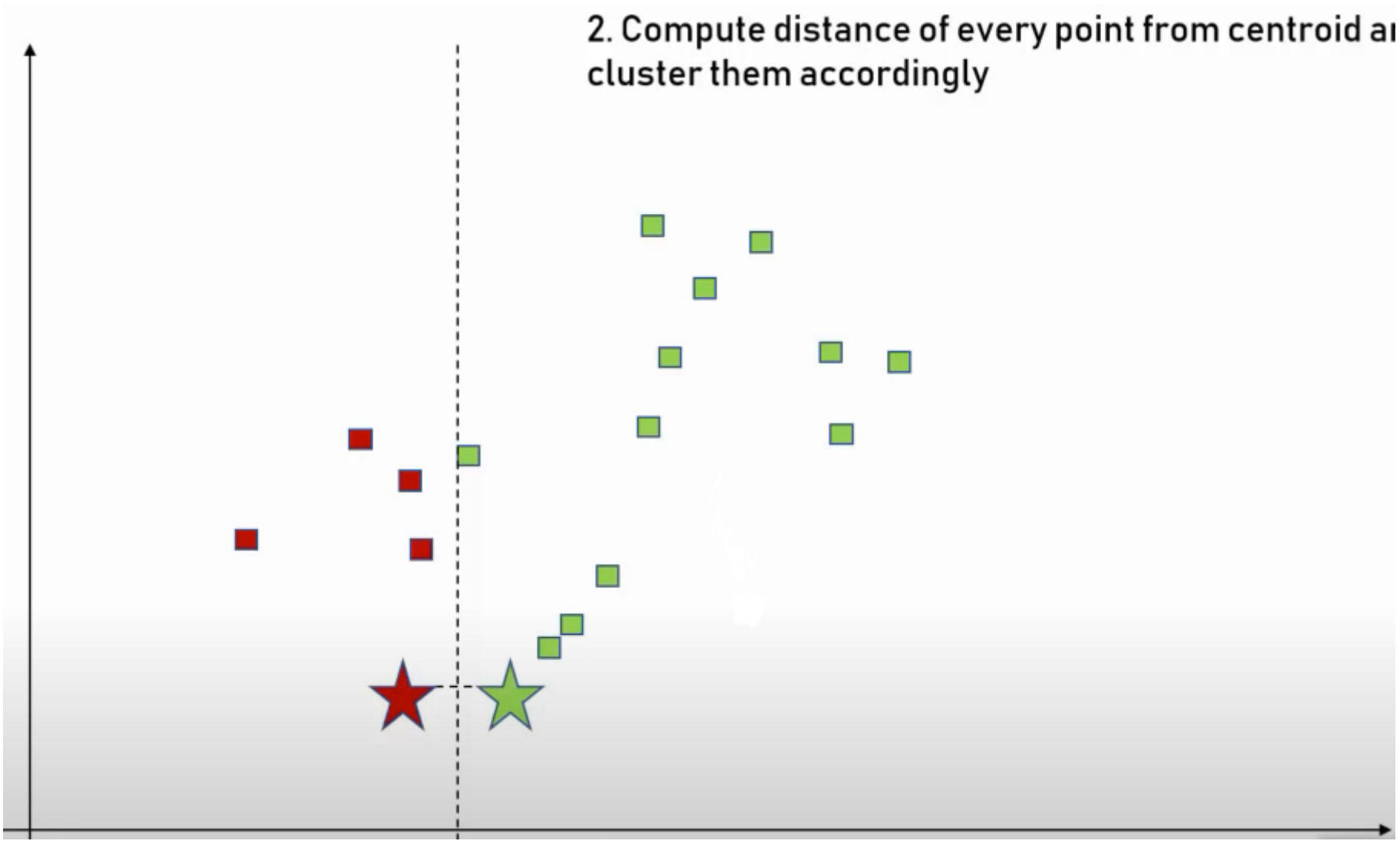


The next step is calculating the distance of each data point to the centroids, and the nearest centroid determines to which cluster a data point is part of. For the distance, the Euclidean distance is considered.

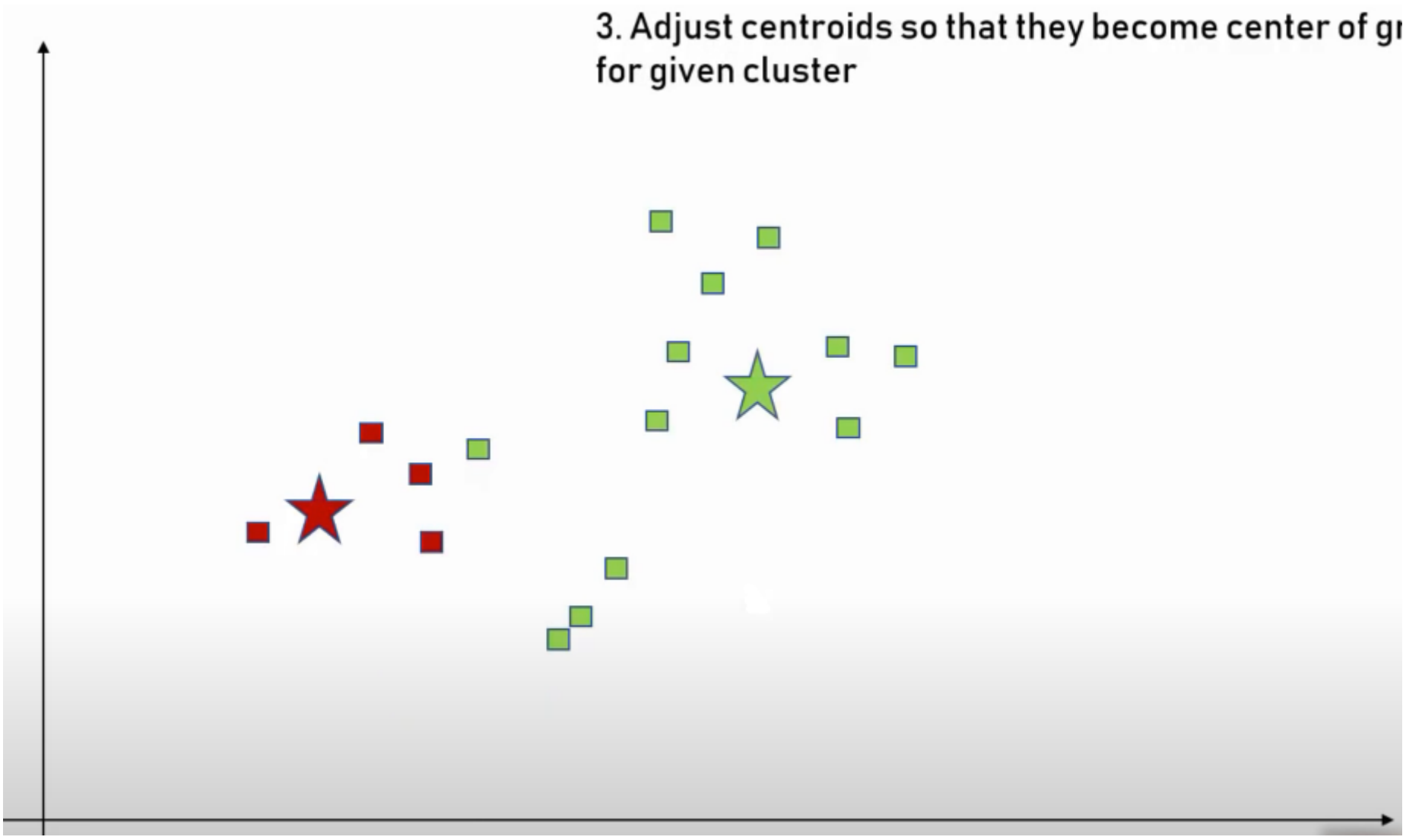
The euclidean distance formula is as follows:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}$$

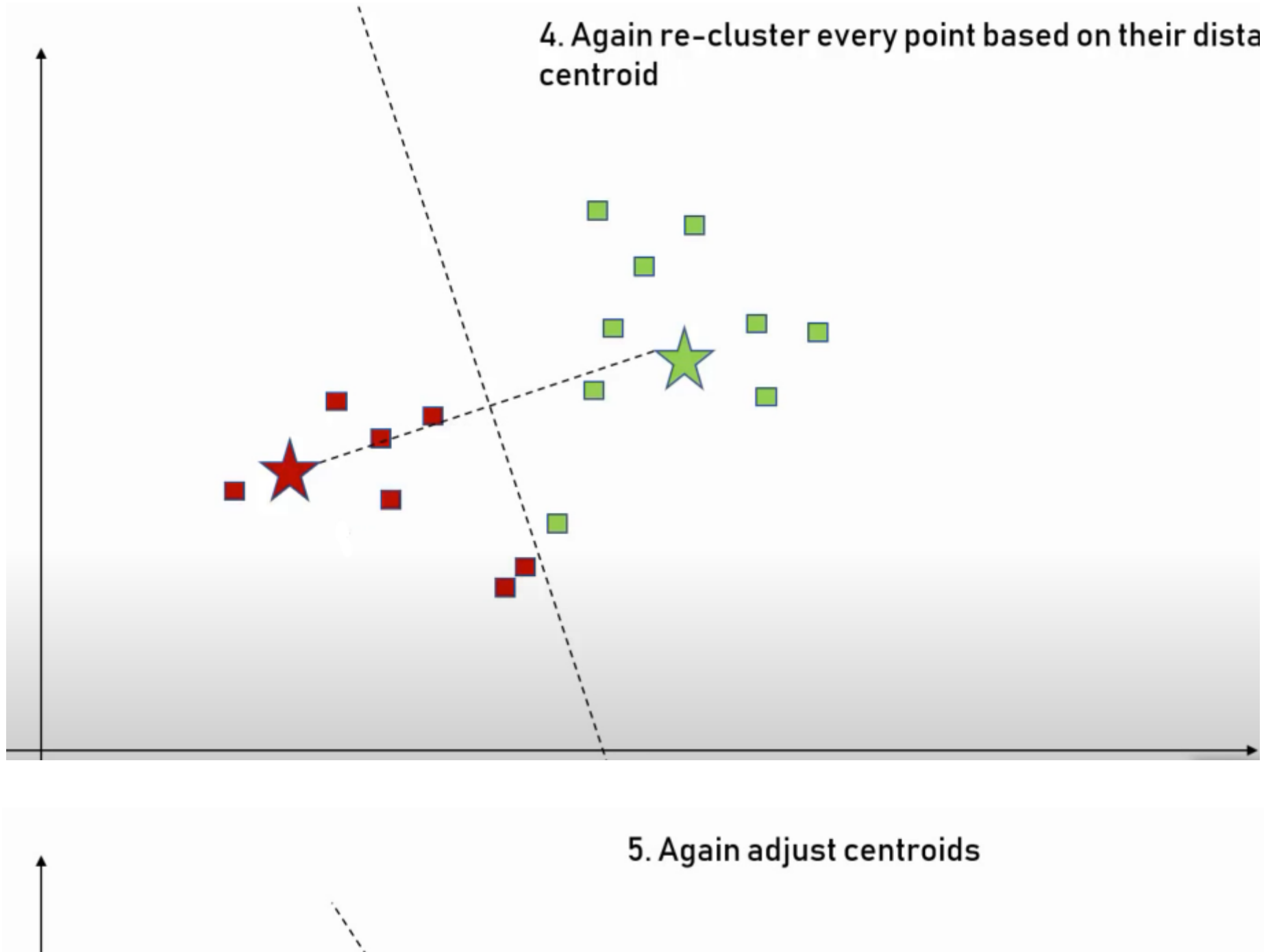
With all the distances calculated and all the data points clustered accordingly (the red data points belong to the cluster with the red star as centroid, and the green data points belong to the other cluster with the green star as centroid), the dataset will look like this:

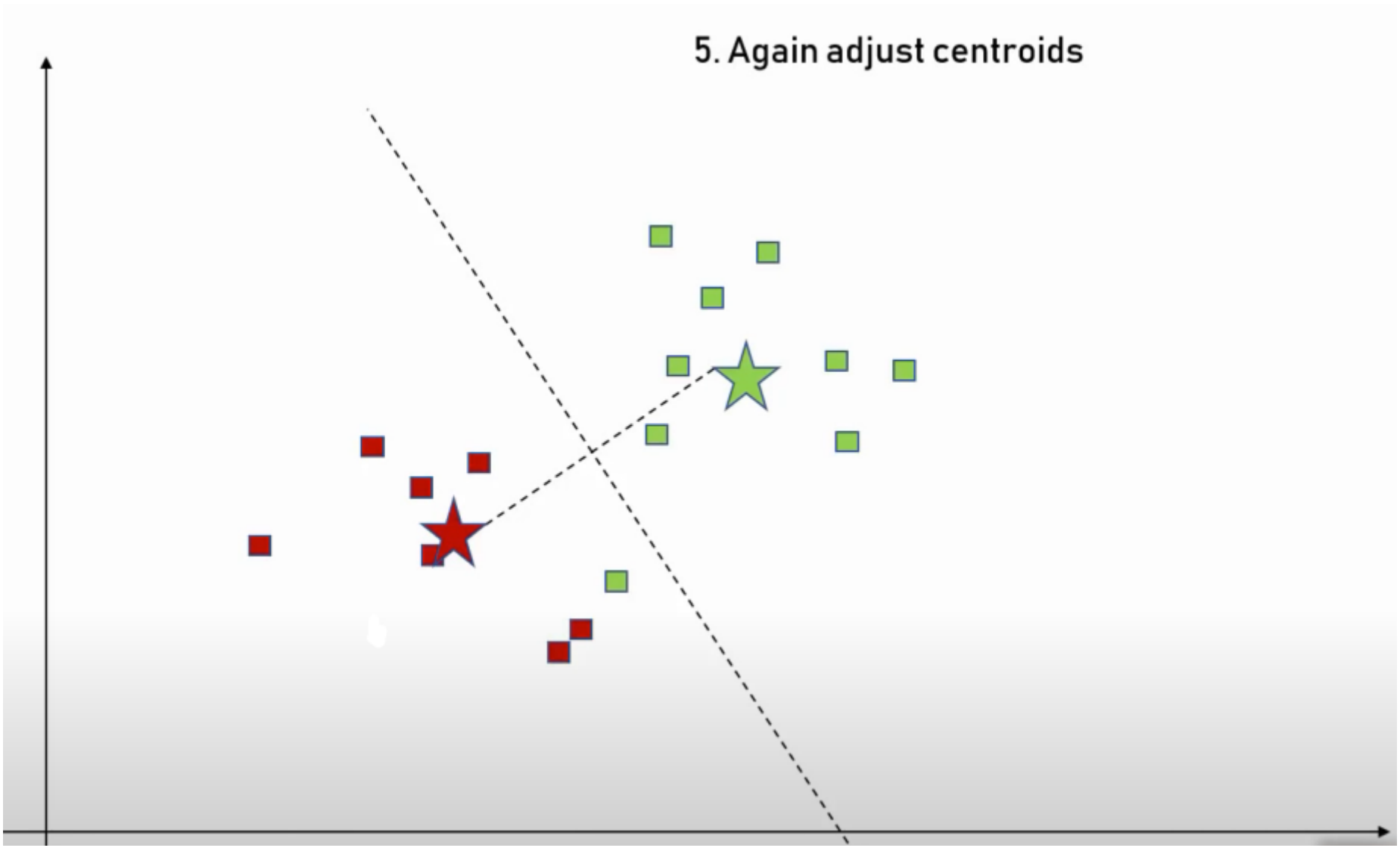


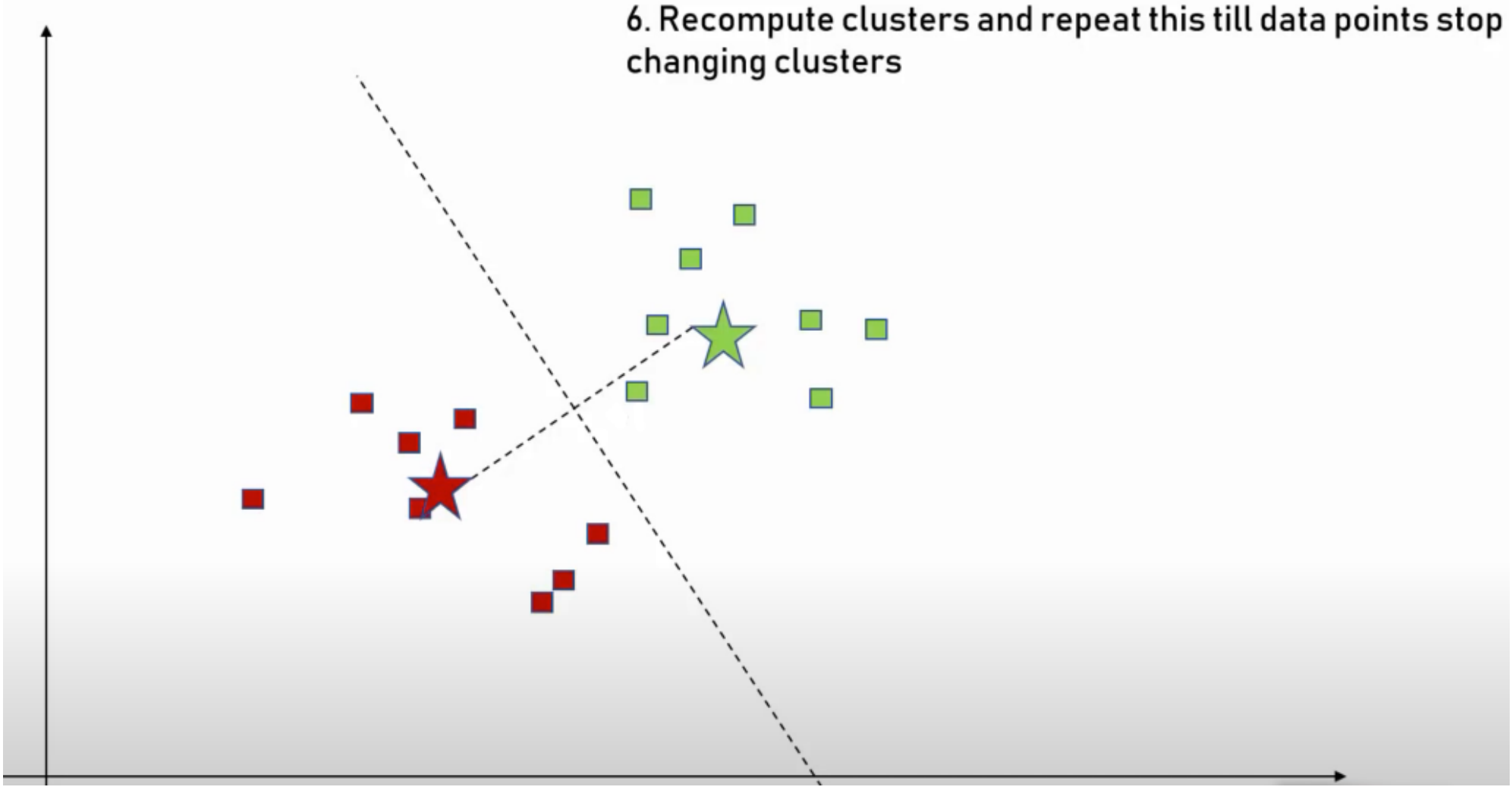
Henceforth, the algorithm will iteratively reposition the centroids by calculating the means of the coordinates of the data points in the cluster. In this example, the coordinates for a new cluster centroid will be the mean of the x-axis and y-axis coordinates of the data points in that cluster. That process will repeat itself until the centroids are stabilized, when even if more iterations are done, the centroids don't change.



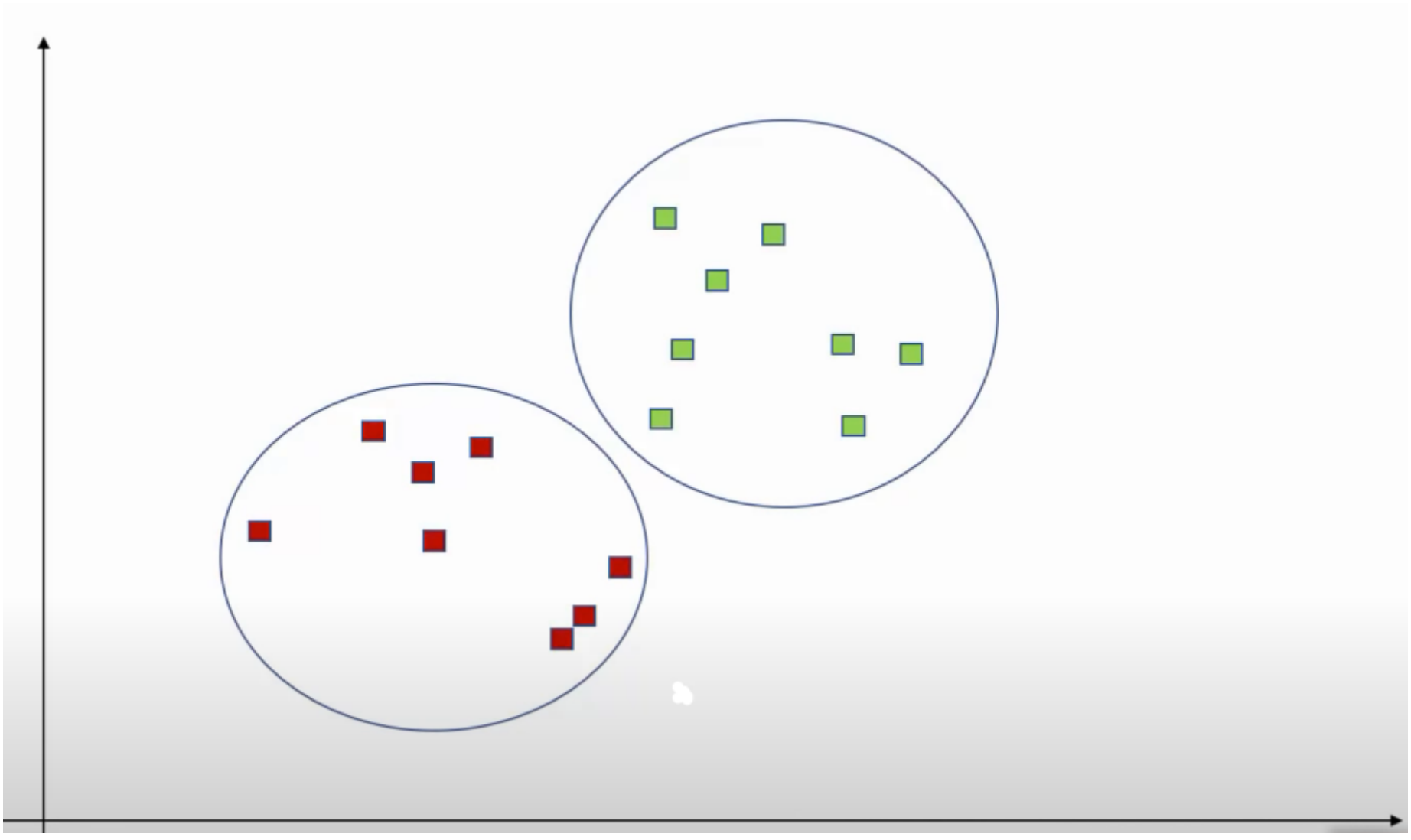
Since the centroids have been repositioned, it's common for some data points to be wrongly clustered, and that's why they must be clustered again, as the distance between them and the centroids have been altered. All the remaining iterations are below.


4. Again re-cluster every point based on their dista
centroid


5. Again adjust centroids



And these are the final clusters once the algorithm stops:



However, the value "k" (the number of clusters) was arbitrarily defined as 2. And that isn't necessarily the ideal value for "k" in this scenario. Therefore, in order to find the best value for "k", the elbow method can be used. This method consists of plotting the graph WCSS (Within-Cluster Sum of Squares) vs. "k", and picking the elbow of the curve as the value of "k".

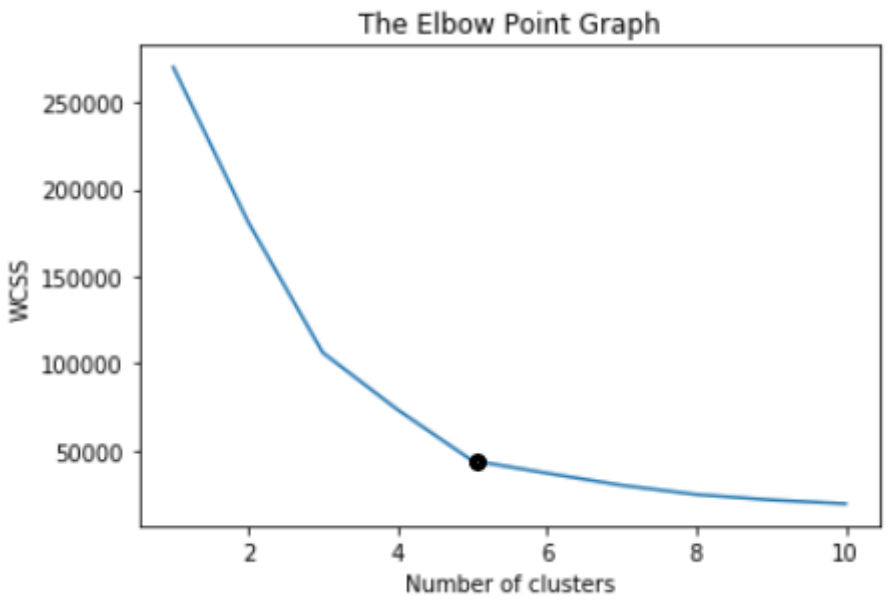
SSE can be used as a measure of variation within a cluster; the closer the data points are to the centroid, the closer the SSE will be to 0. The equation for the SSE is as follows (the 1 after the SSE indicates that it only considers one cluster in the calculation):

$$SSE_1 = \sum_{i=0}^{n} dist(x_i - c_1)^2$$

Moreover, the WCSS, taking into account all of the clusters, is the sum of the SSE of each cluster.

$$WCSS = \sum_{i=1}^{k} SSE_i$$

The WCSS decreases as the number of clusters increases, as a result of the data points becoming closer to the centroids in each cluster. Still, the number of clusters shouldn't unnecessarily large, because that'd make the cluster more exclusive than they should. For that reason, the ideal number of "k" is the elbow of the curve on the graph below, since it's the point where the variation of the WCSS starts to decrease. The graph below is just an example, it isn't related to the examples above.



With these, Machine Learning algorithms can make predictions.

K-means Clustering Algorithm in python

Before coding the algorithm, some libraries must be imported:

```
import plotly.express as px
import plotly.graph_objects as go
import numpy as np
from sklearn.preprocessing import StandardScaler
```

Now, K-means must be imported as well.

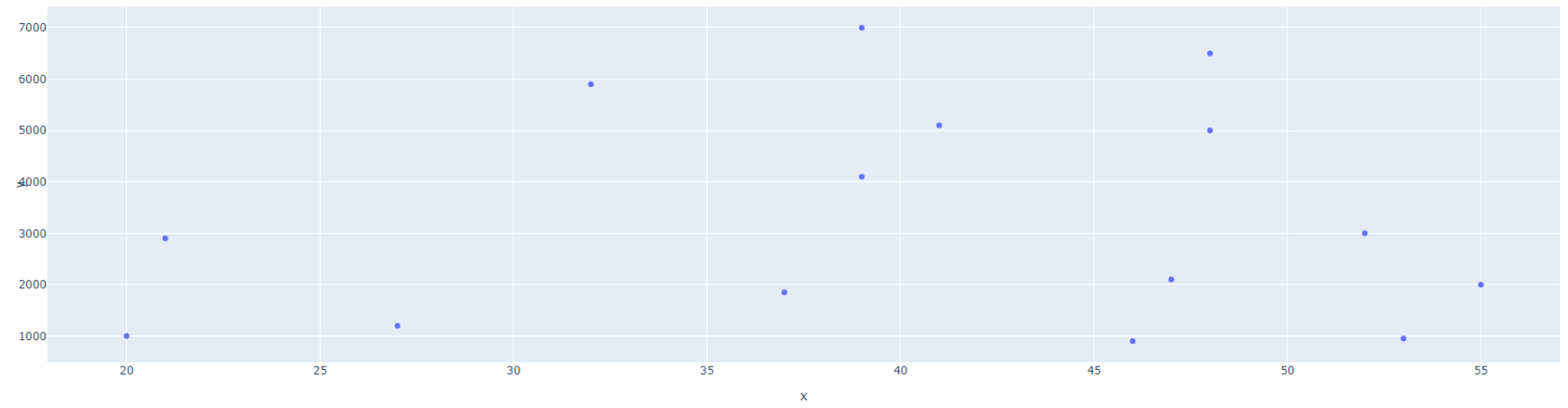
```
from sklearn.cluster import KMeans
```

The arrays below represent the data points (age and salary of some people used in this example) that'll be clustered, a graph can be plotted with the coordinates.

```
x=[20,  27,  21,  37,  46, 53, 55,  47,  52,  32,  39,  41,  39,  48,  48]
y=[1000, 1200, 2900, 1850, 900, 950, 2000, 2100, 3000, 5900, 4100, 5100, 7000, 5000, 6500]
```

```
graph = px.scatter(x = x, y = y)
graph.show()
```

output:



Now, through the numpy library, the arrays will be put together in a matrix.

```
salary_base  = np.array([[20,1000],[27,1200],[21,2900],[37,1850],[46,900],
                        [53,950],[55,2000],[47,2100],[52,3000],[32,5900],
                        [39,4100],[41,5100],[39,7000],[48,5000],[48,6500]])
```

Since the data is not scaled, it must be scaled to be properly used in a Machine Learning algorithm.

```
scaler_salary = StandardScaler()
salary_base = scaler_salary.fit_transform(salary_base)
print(salart_base)
```

output:

```
array([[-1.87963884, -1.11413572],
       [-1.23255006, -1.01725435],
       [-1.78719758, -0.19376273],
       [-0.30813751, -0.70238991],
       [ 0.52383377, -1.1625764 ],
       [ 1.17092255, -1.13835606],
       [ 1.35580506, -0.62972888],
       [ 0.61627503, -0.5812882 ],
       [ 1.0784813 , -0.14532205],
       [-0.77034379,  1.25945777],
       [-0.12325501,  0.38752547],
       [ 0.0616275 ,  0.8719323 ],
       [-0.12325501,  1.79230528],
       [ 0.70871628,  0.82349162],
       [ 0.70871628,  1.55010187]])
```

With the data scaled, lets start the K-Means algorithm. In this example, k is defined as 3.

```
kmeans_salary = KMeans(n_clusters=3)
kmeans_salary.fit(salary_base)
```

The coordinates of the clusters can be found with "kmeans.cluster_centers_" function.

scaled coordinates:

```
centroids = kmeans_salary.cluster_centers_
print(centroids)
```

output:

```
array([[ 0.73953003, -0.72661025],
       [ 0.07703438,  1.11413572],
       [-1.63312883, -0.77505093]])
```

nonscaled coordinates:

```
print(scaler_salary.inverse_transform(kmeans_salary.cluster_centers_))
```

output:

```
array([[  48.33333333, 1800],
       [  41.16666667, 5600],
       [  22.66666667, 1700]])
```

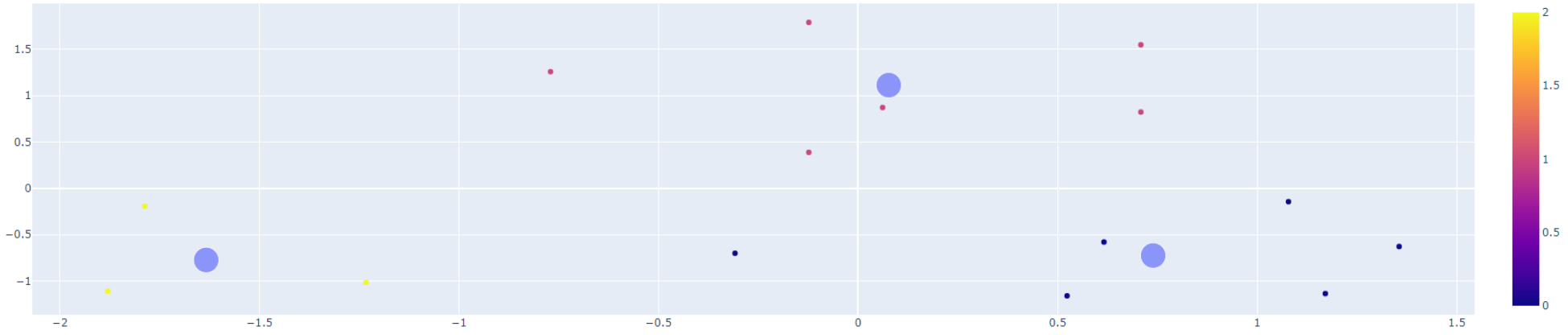
Finally, the clusters can be visualized.

```
labels = kmeans_salary.labels_

graph_data_points = px.scatter(x = salary_base[:,0], y = salary_base[:,1], color = labels)
graph_centroids = px.scatter(x = centroids[:,0], y = centroids[:,1], size = [12, 12, 12])
graph_clusters= go.Figure(data = graph_data_points.data + graph_centroids.data)

graph_clusters.show()
```

output:


1.5
1
0.5
0
−0.5
−1
−2
−1.5
−1
−0.5
0
0.5
1
1.5
2
1.5
1
0.5
0