Clustering

Shivam Kalra & Aditya Sriram

University of Waterloo

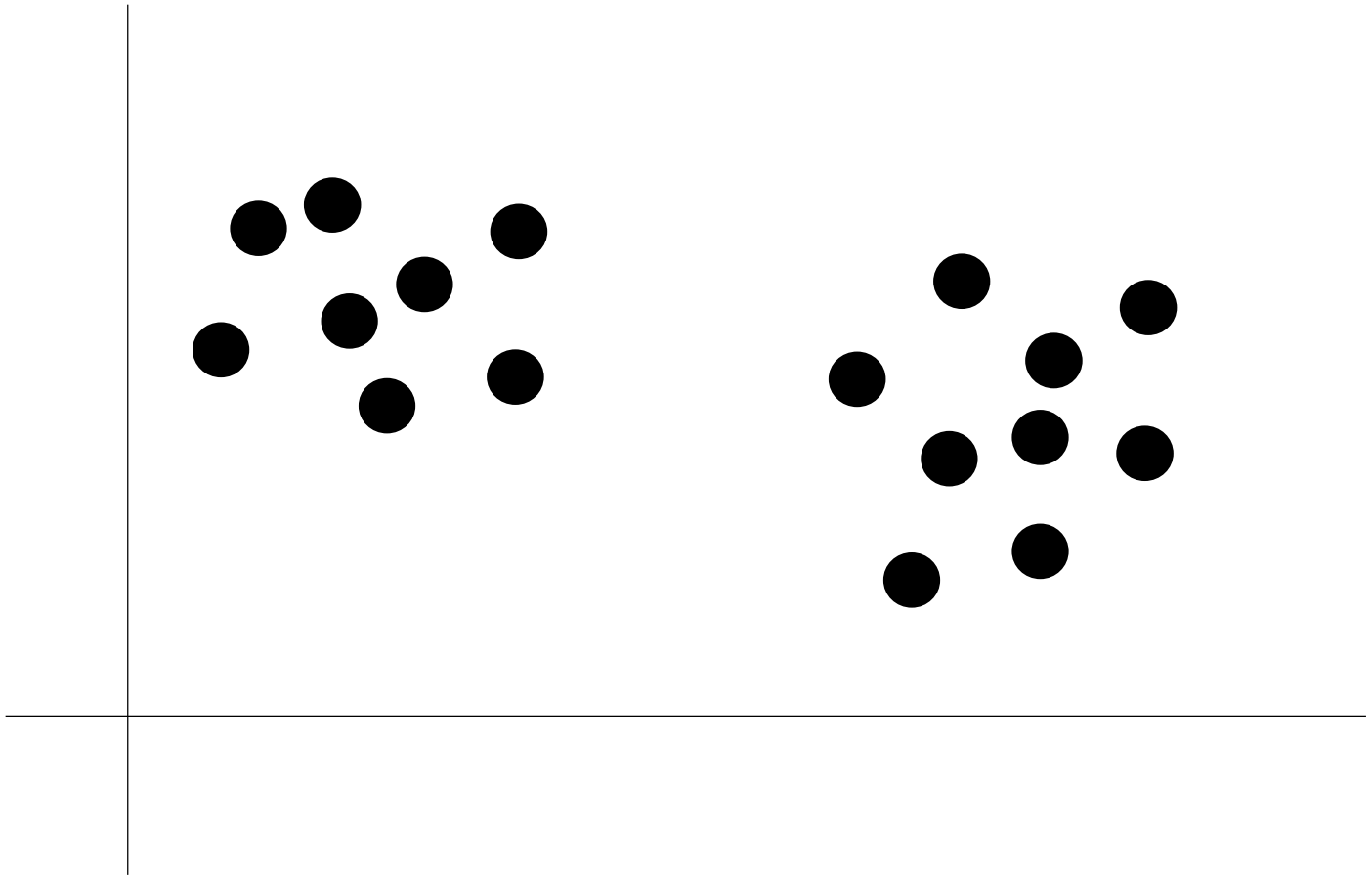
Today Topics

- Clustering (Shivam)
- K-means clustering (Shivam)
- Self-organizing map (Aditya)

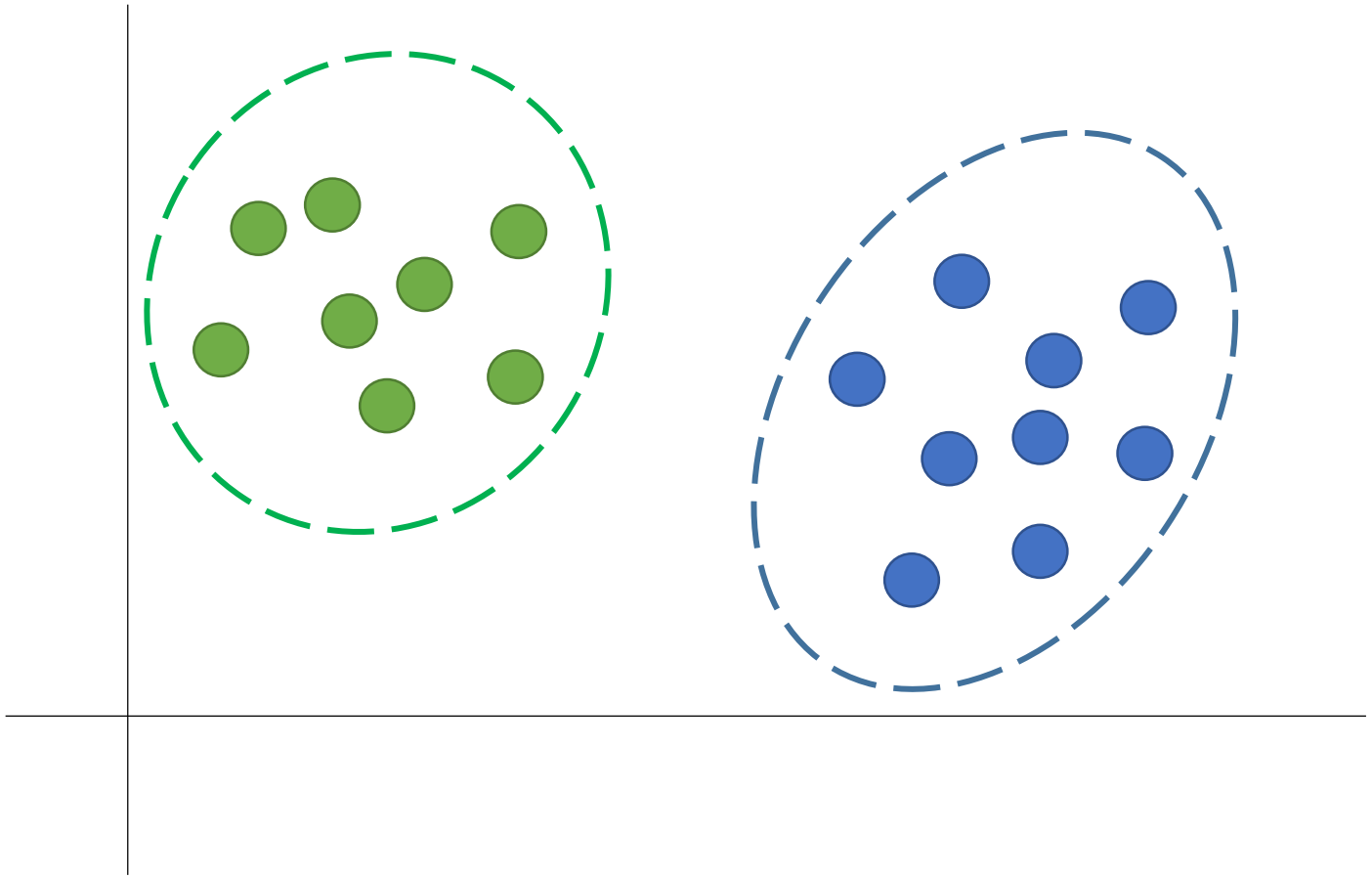
Unsupervised learning

- In supervised learning, data is $< x, y >$ where $y = f(x)$ and goal is to predict f.
- In ***unsupervised learning***, data is $\boldsymbol{x}$!
- Goal is to find “patterns” or “groups” within data.
- Example:
  - Image segmentation
  - Recommender systems
  - Anomaly detection

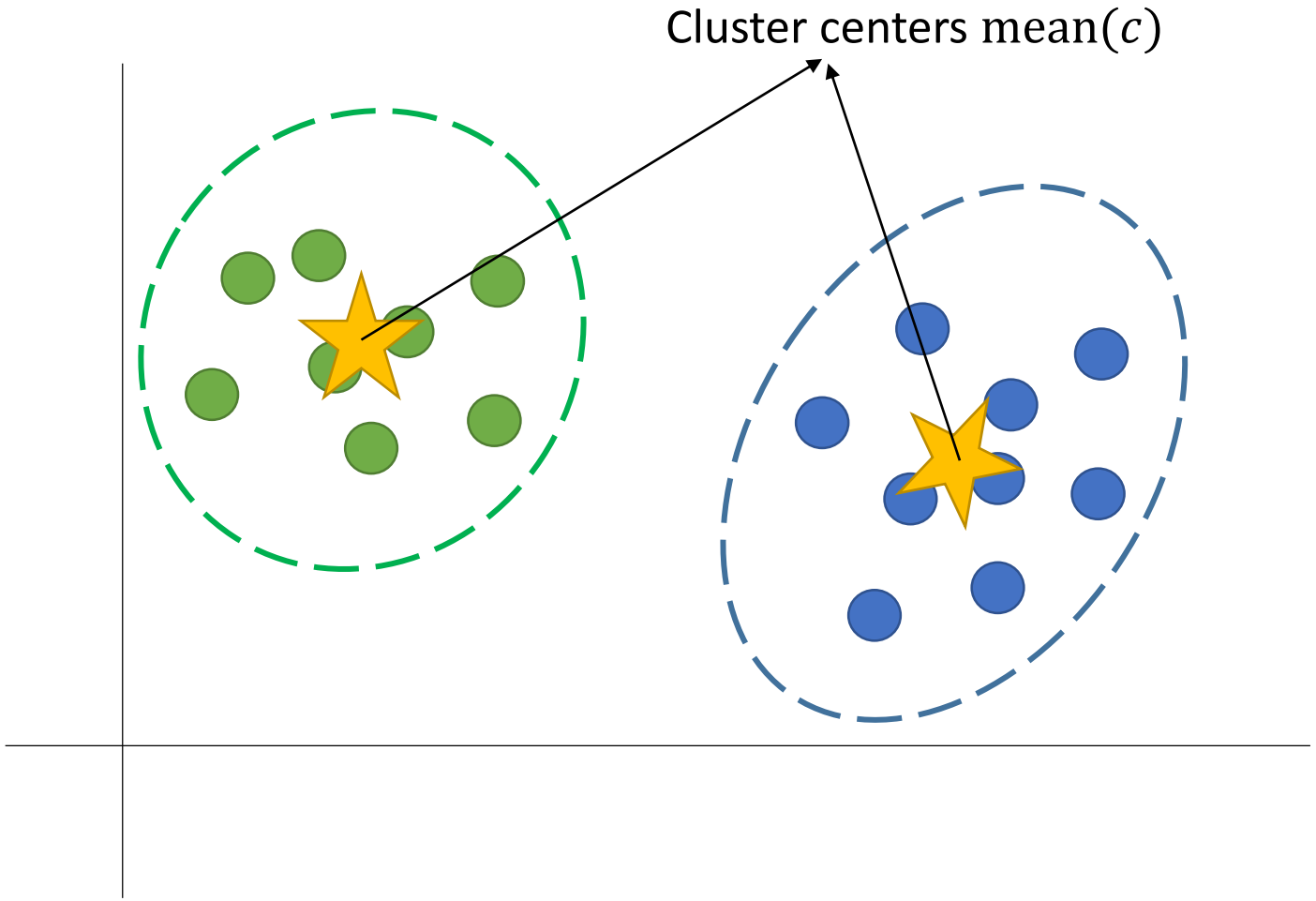
Clustering



Clustering



Clustering

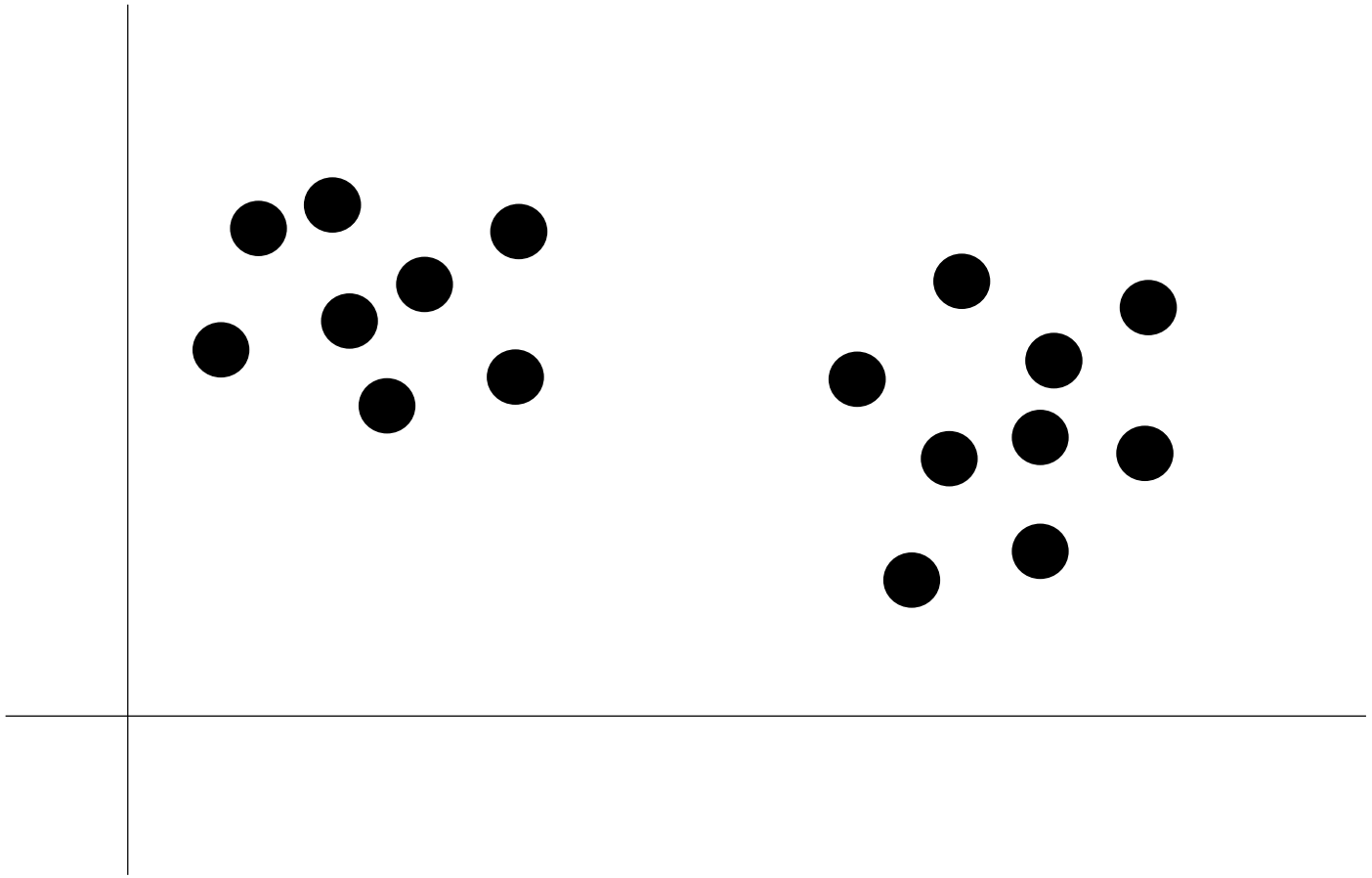


Clustering as *optimization*

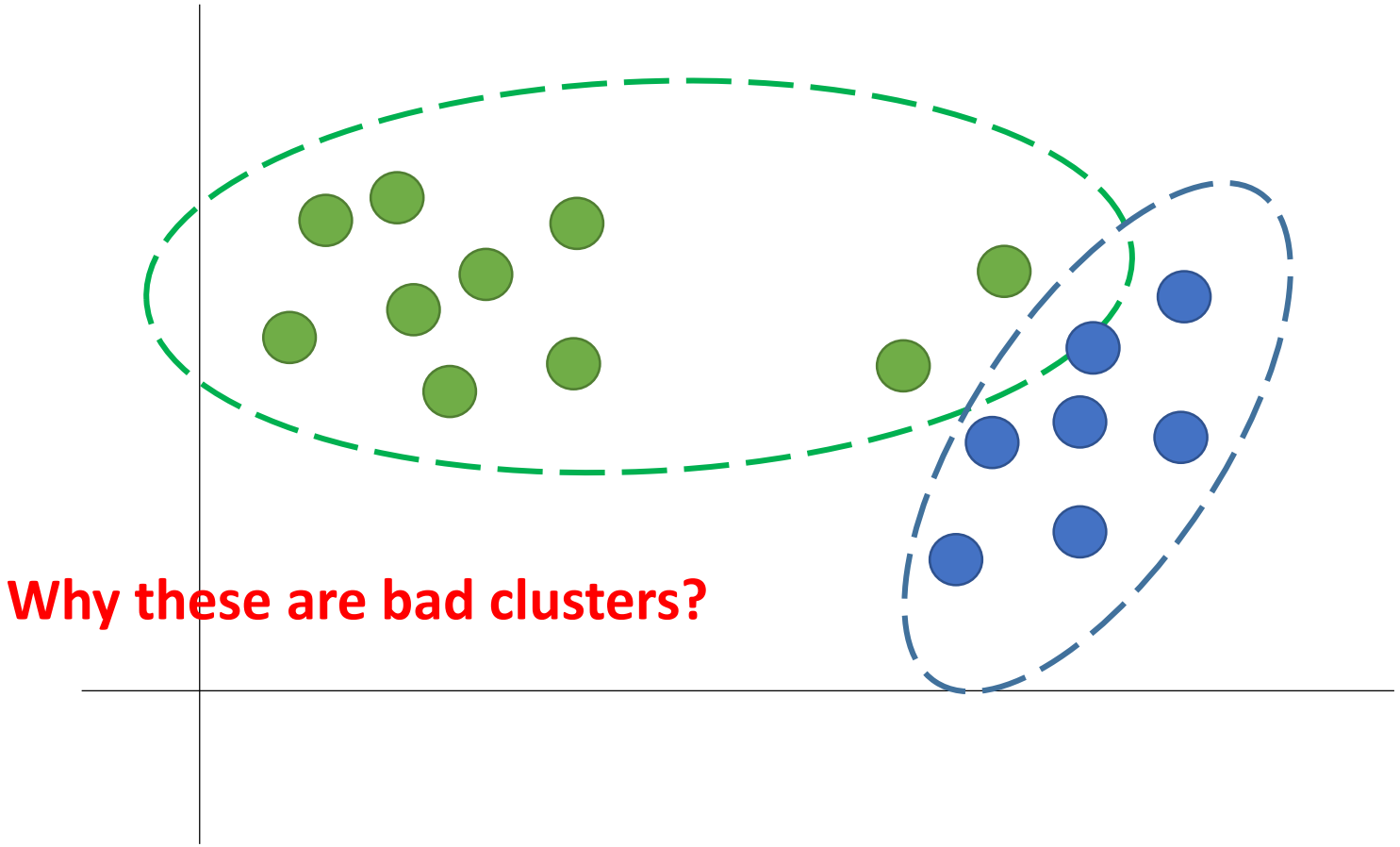
- Formulate clustering as optimization problem

$$\arg\min_{C} f_{obj}(C = \{c_1, c_2, \ldots, c_n\})$$

Clustering as *optimization*

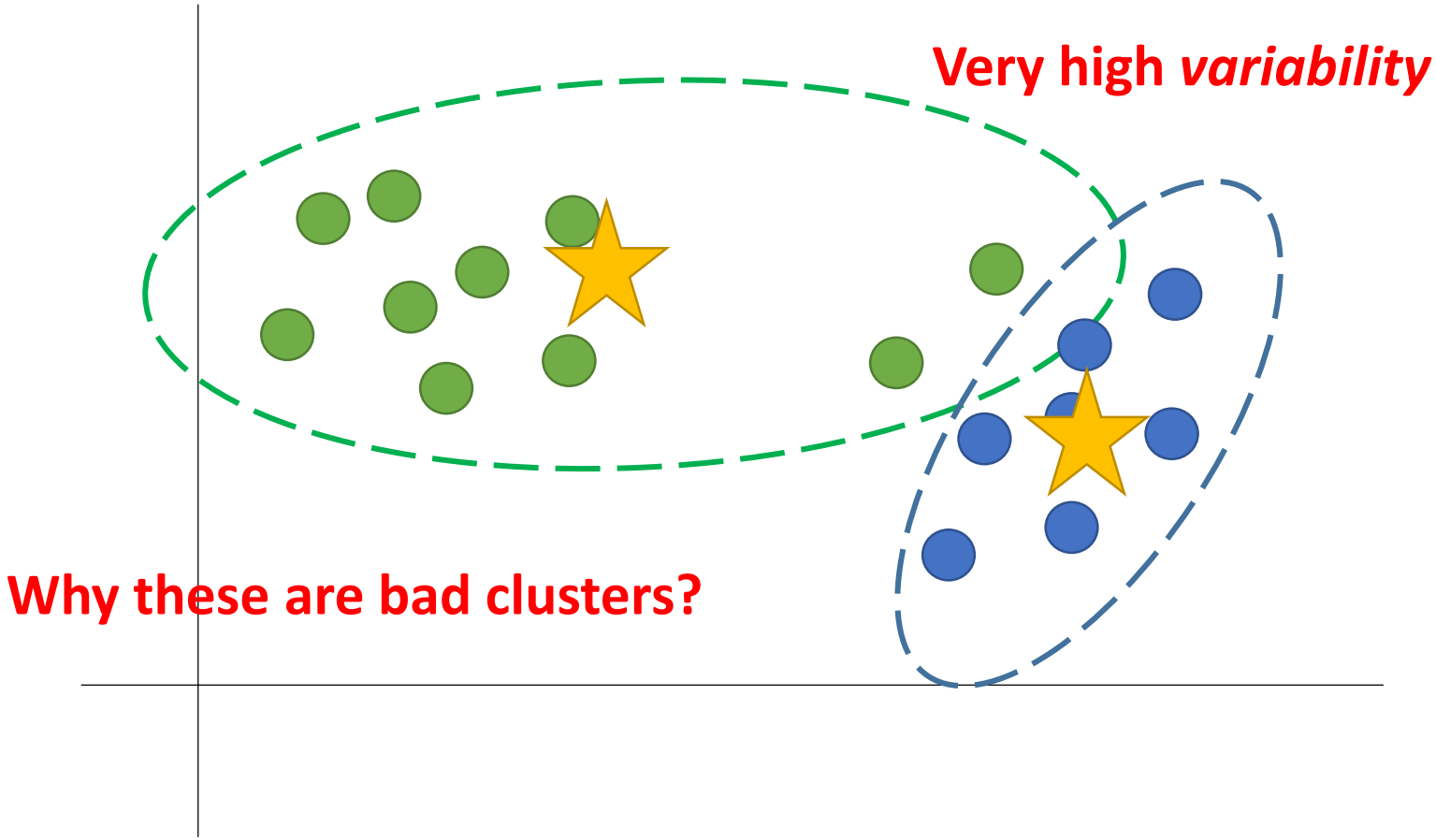


Clustering as *optimization*



Why these are bad clusters?

Clustering as *optimization*



Clustering as *optimization*

$$variability(c) = \sum_{e \in c} distance(mean(c), e)^2$$

f_{obj}

$$dissimilarity(C) = \sum_{c \in C} variability(c)$$

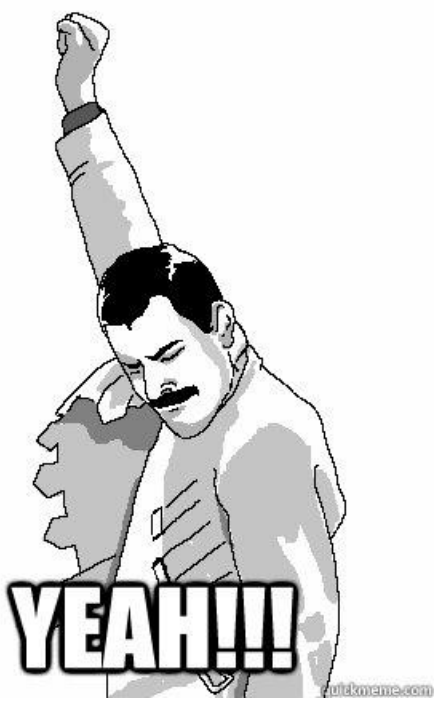
Clustering as *optimization*

- Clustering is minimization of $dissimilarity(C)$
- Constraint: $|C| < n$ where n are number of data-points

K-means Algorithm

```
1. randomly choose k random cluster centers
2. while  true:
3.      create k cluster
4.             according to distance from center
5.      compute k new centers
6.             mean of elements in each cluster
7.      if centers don't change
8.             break
```

K-means Algorithm (one line)




```
KMeansFit = lambda X, centers, max_itr, k: [[np.mean(X[np.argmin([np.linalg.norm(X-m_r, axis=1) for m_r in centers], axis=0) == r], axis=0) for r in range(k)] for itr in range(max_itr)]
```

K-means Algorithm (one line)

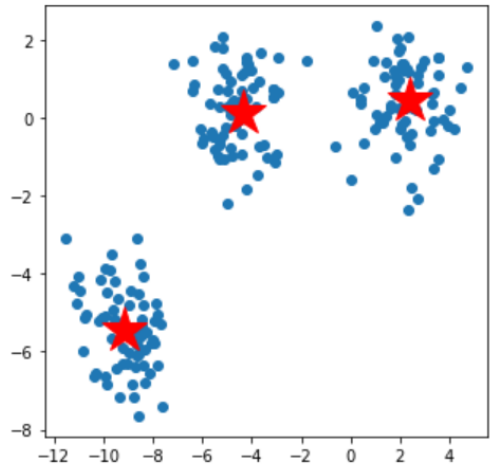
In [2]:
```
import numpy as np
KMeansFit = lambda X, centers, max_itr, k: [[np.mean(X[np.argmin([np.linalg.norm(X-m_r,
axis=1) for m_r in centers], axis=0) == r], axis=0) for r in range(k)] for itr in
range(max_itr)]
```

In [20]:
```
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

%matplotlib inline


n_samples = 200
random_state = 170
X, _ = make_blobs(n_samples=n_samples, random_state=random_state)

plt.figure(figsize=(5, 5))
plt.axis('off')
plt.scatter(X[:, 0], X[:, 1])
```

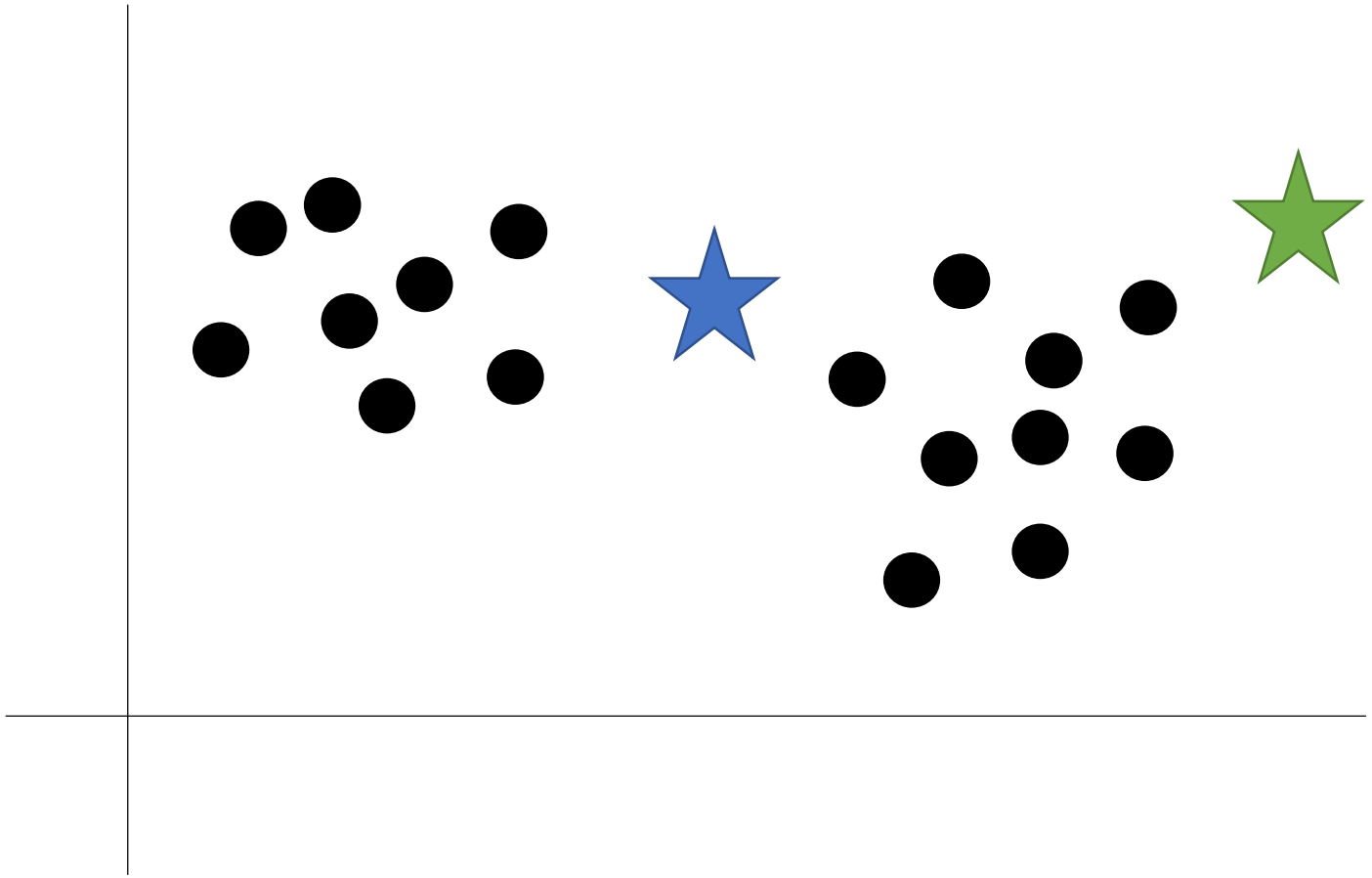
Out[20]: <matplotlib.collections.PathCollection at 0x7f7aa2fcb8d0>

In [18]:
```
c = np.array(KMeansFit(X, [[-6, -8], [-4, 0], [4, 3]], 4, 3)[-1])

plt.figure(figsize=(5, 5))
plt.scatter(X[:, 0], X[:, 1])
plt.plot(c[:, 0], c[:, 1], 'r*', markersize=30)
```

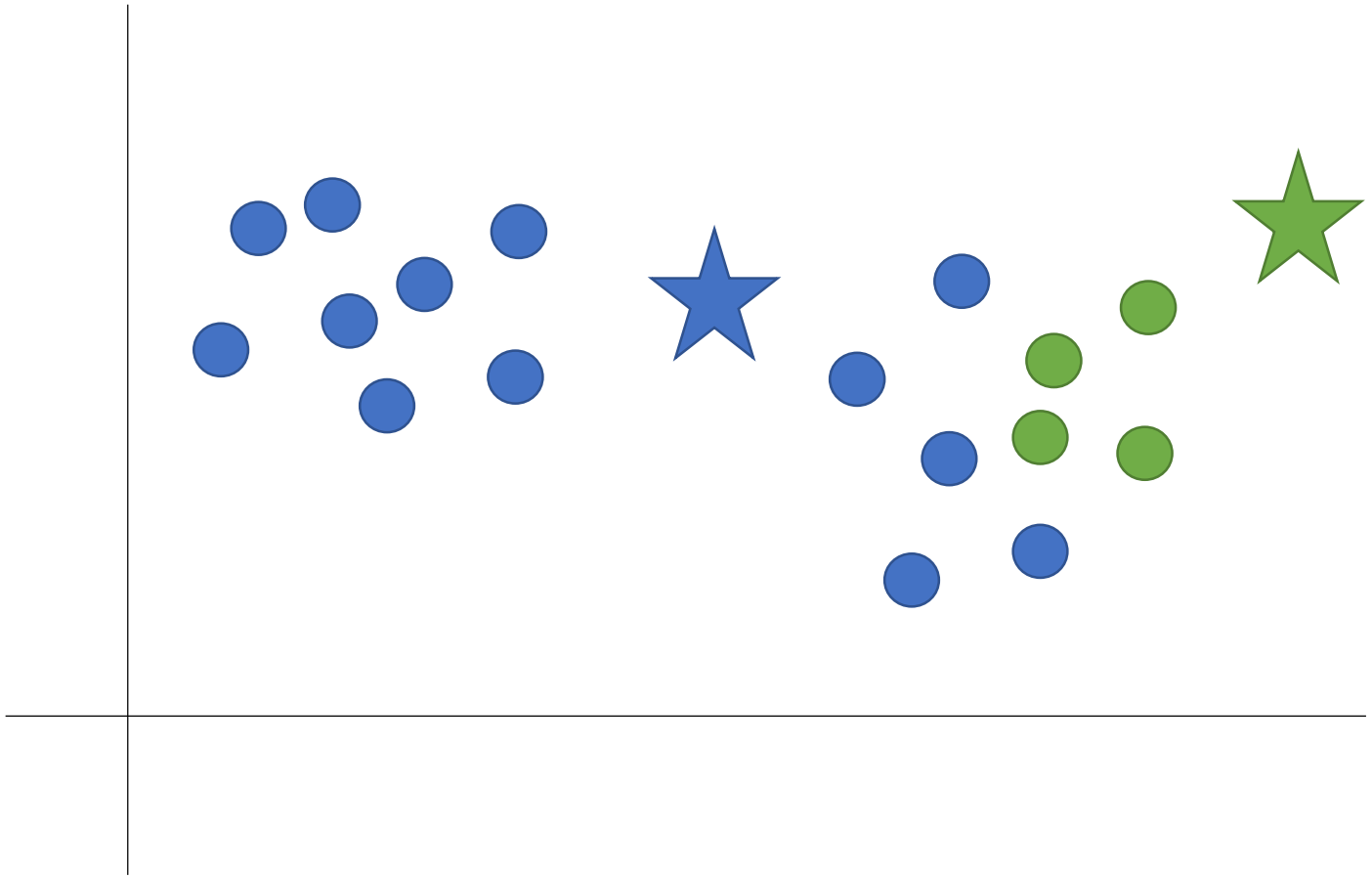
Out[18]: [<matplotlib.lines.Line2D at 0x7f7aa365dfd0>]



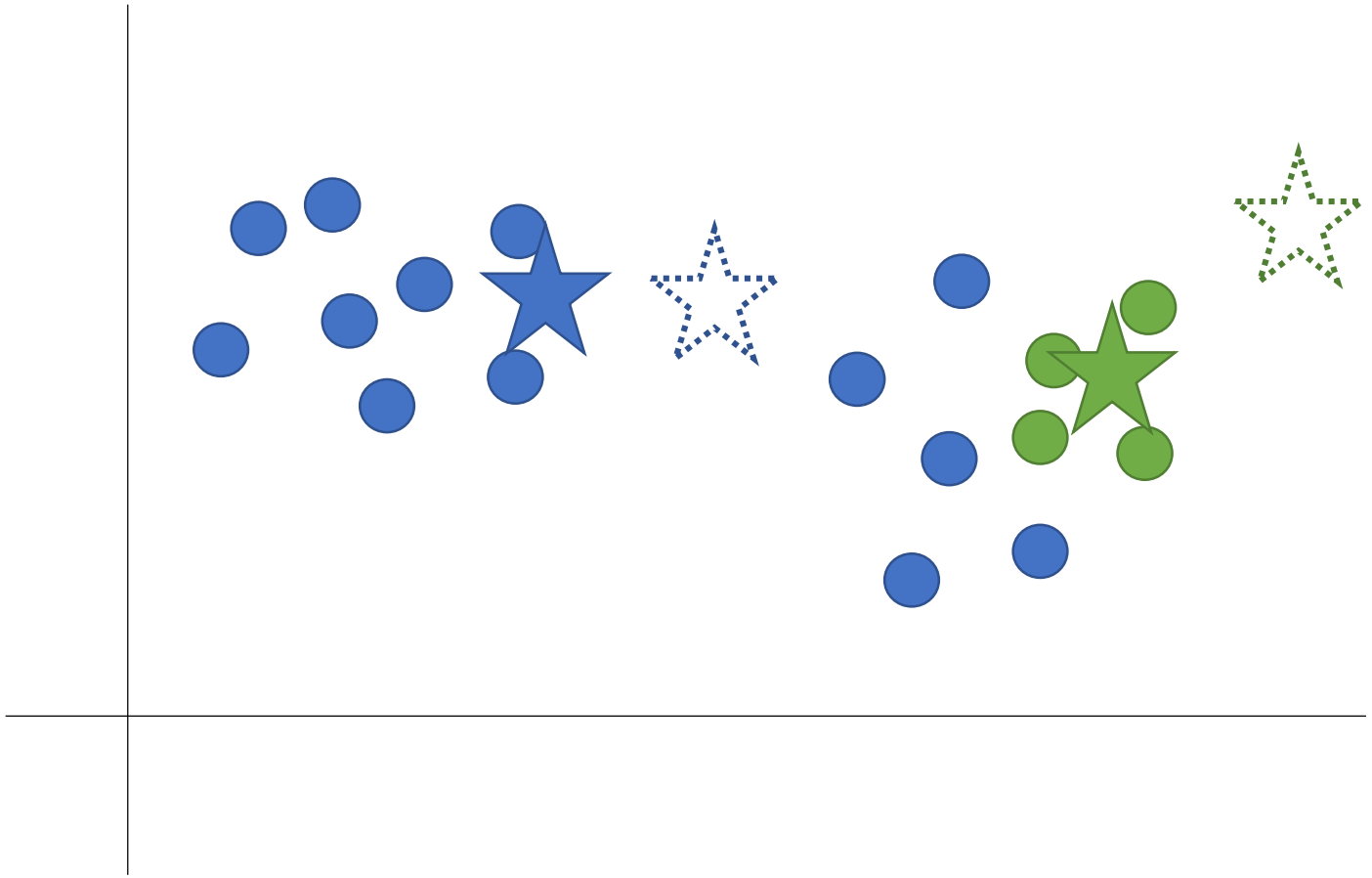
K-means



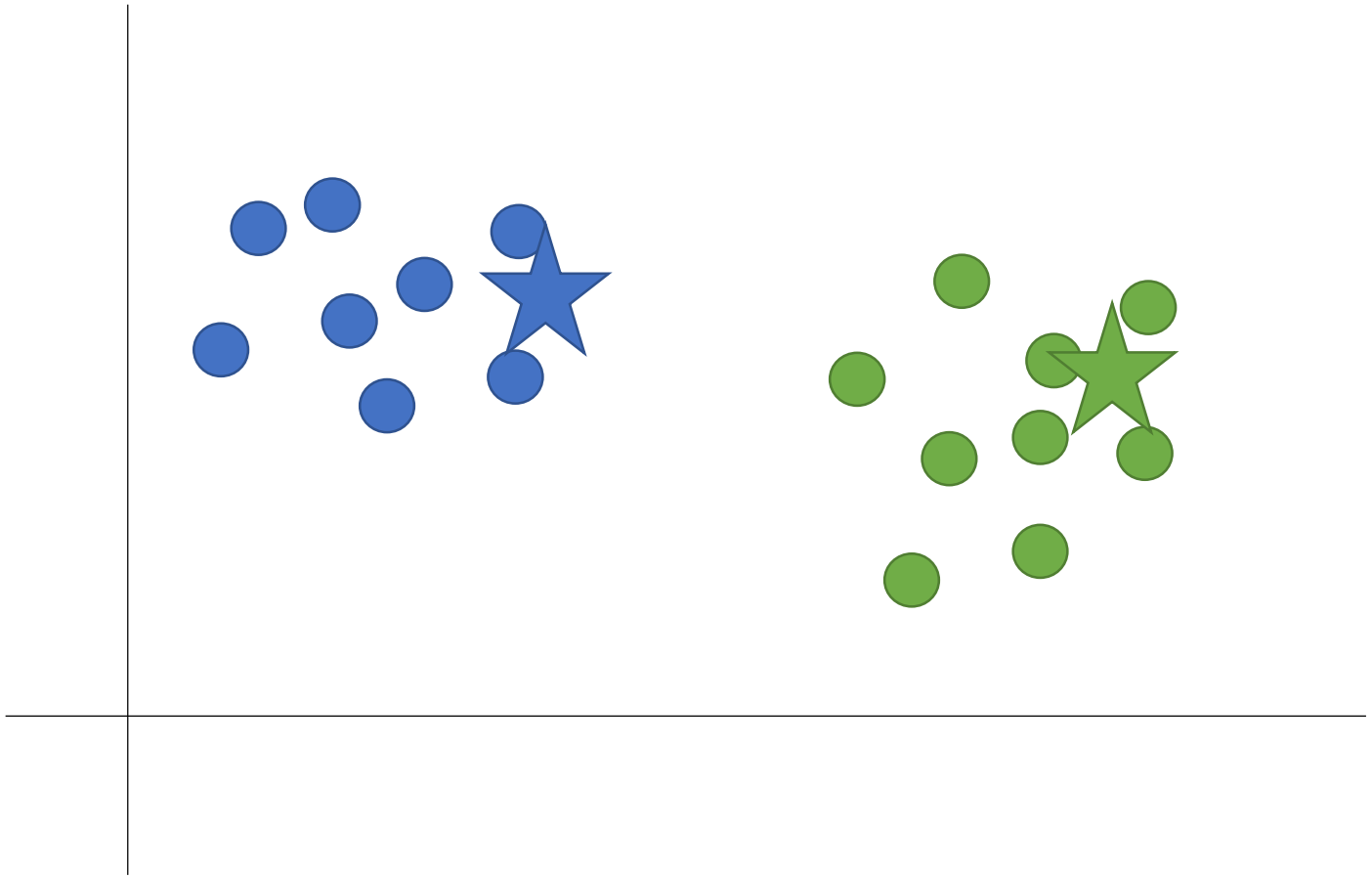
K-means



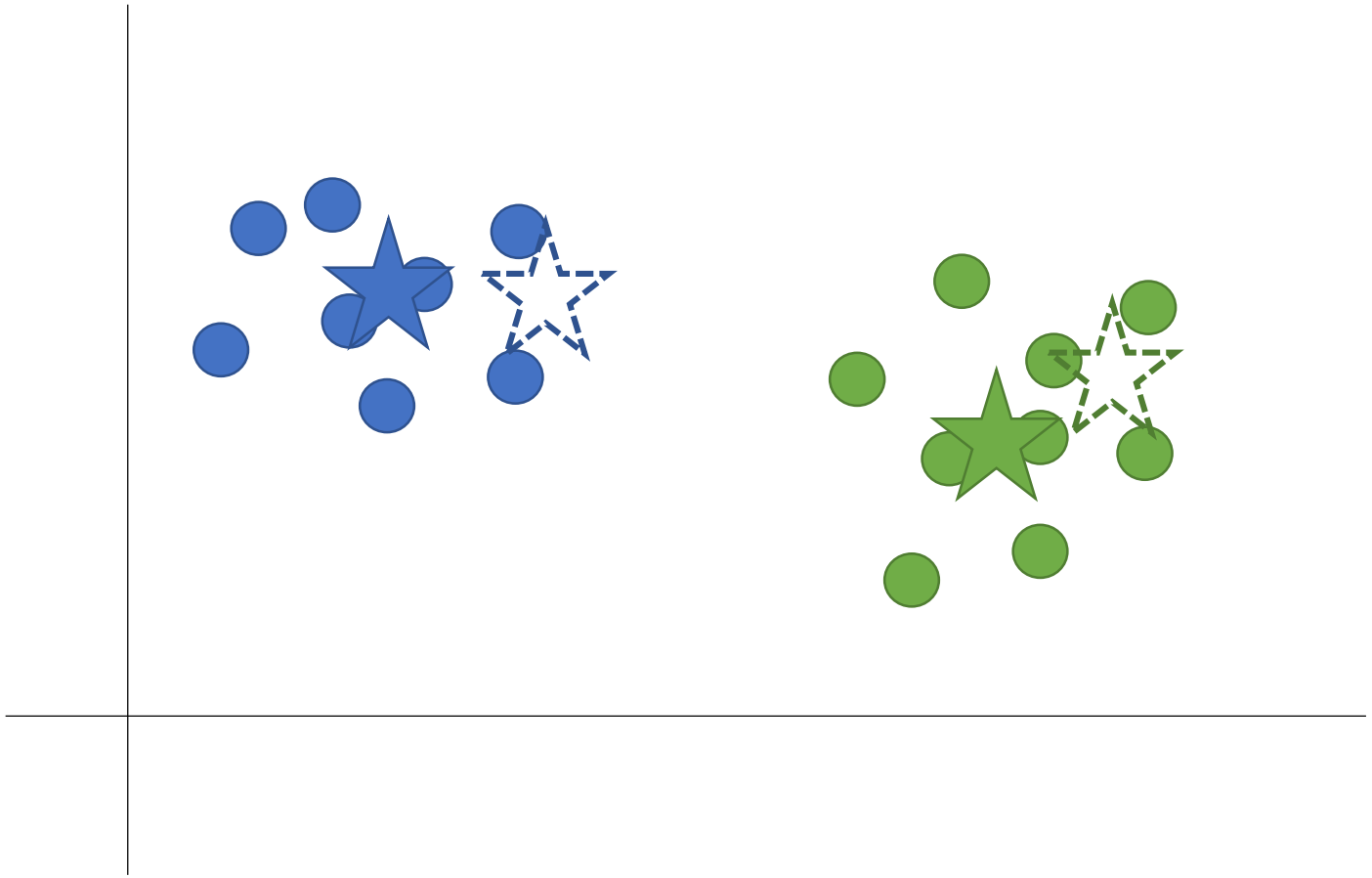
K-means



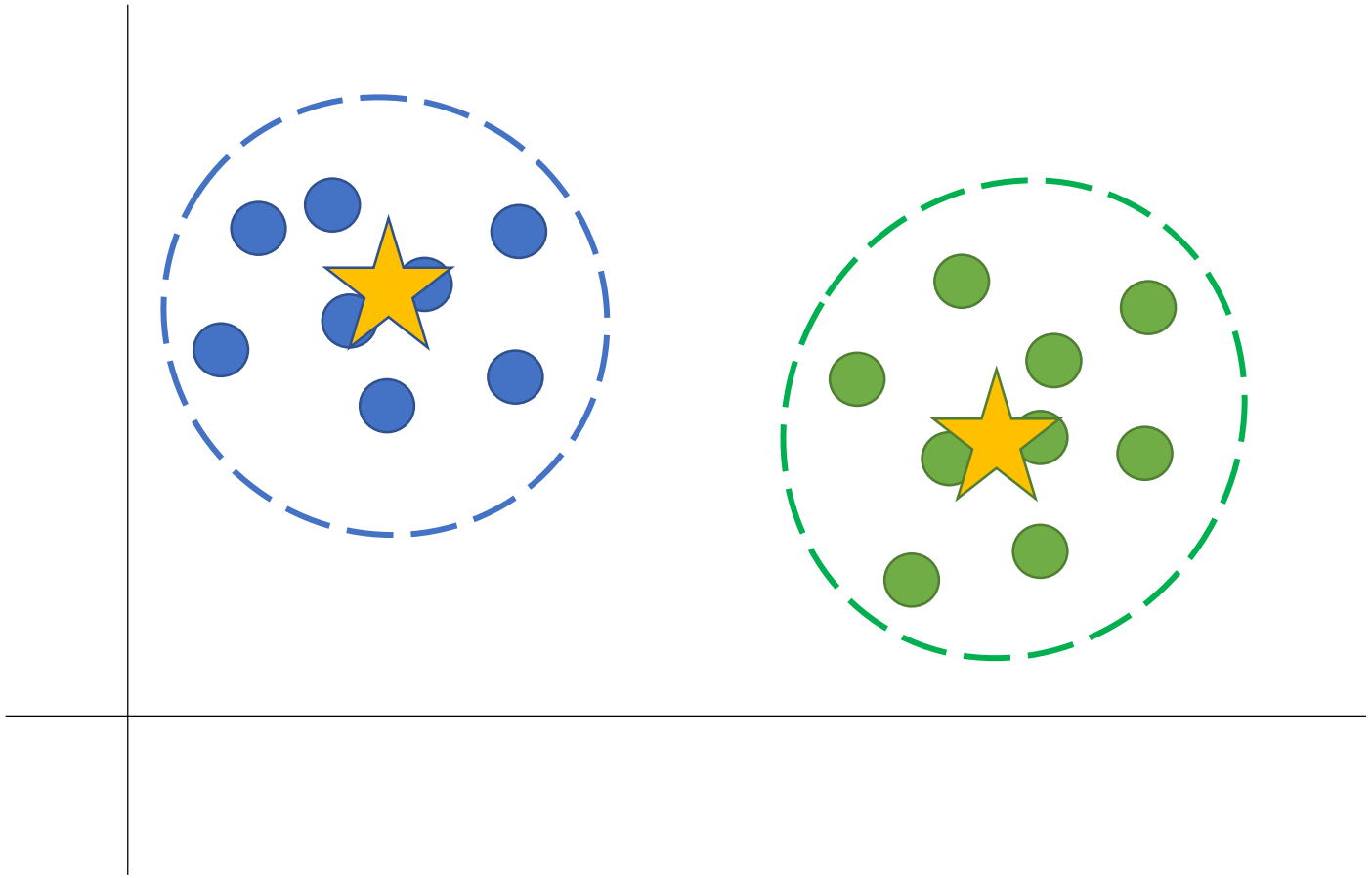
K-means



K-means



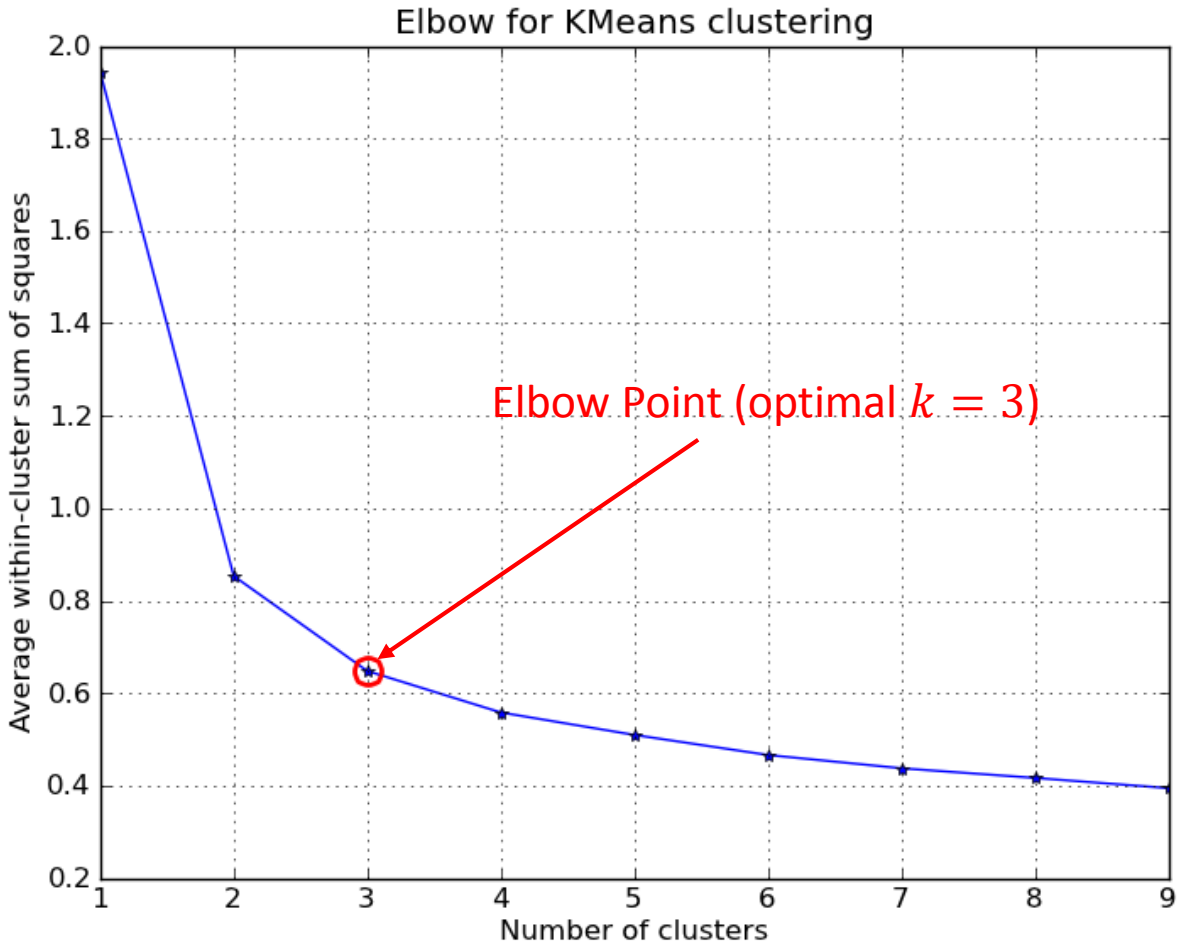
K-means



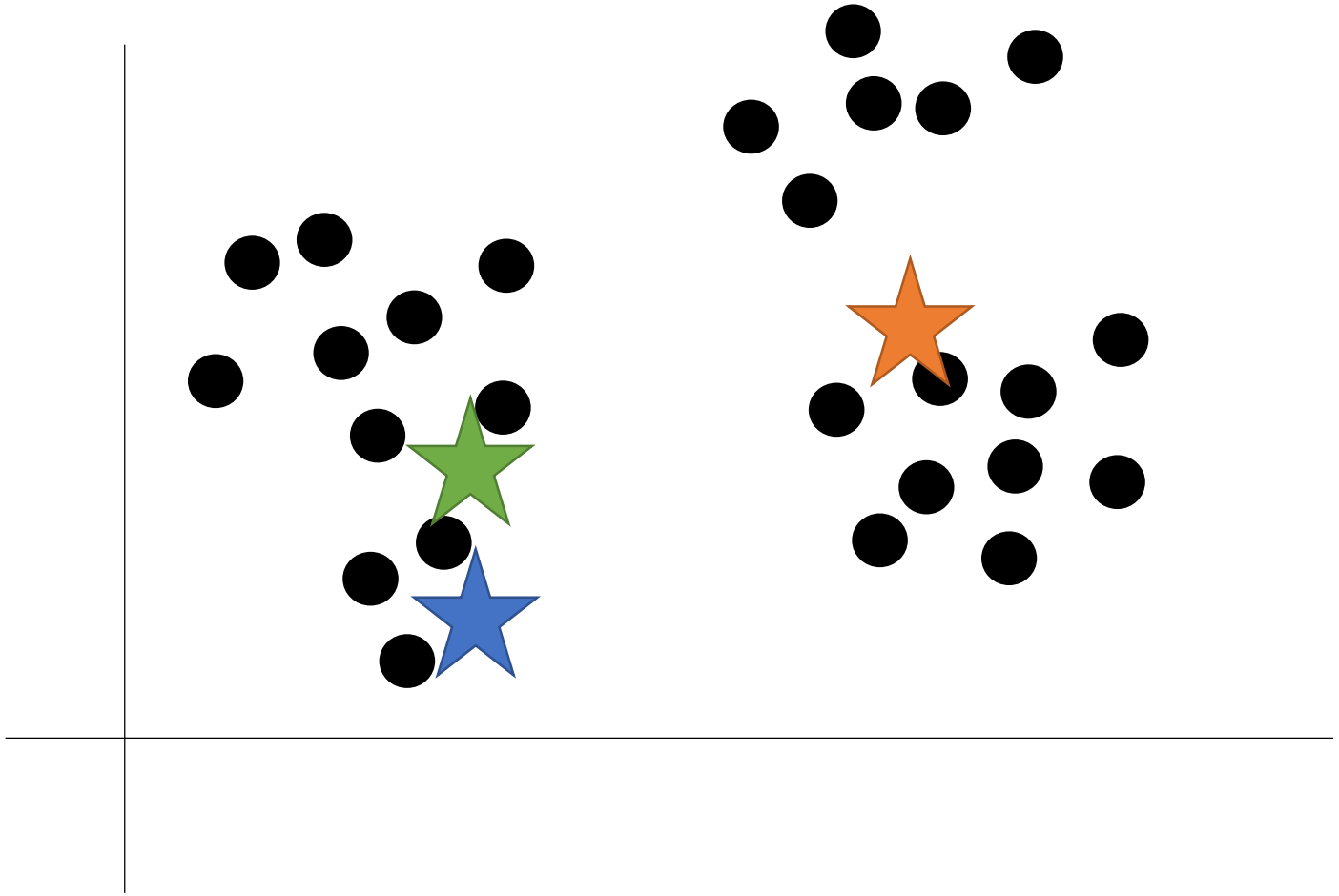
How to choose *k*

- A priori knowledge about the data-points
  - There are kinds of wines: $k = 2$
  - There are two kinds of people: $k = 2$
- Search of a good *k*
  - Try different values of *k* and evaluate quality of results
  - Elbow method

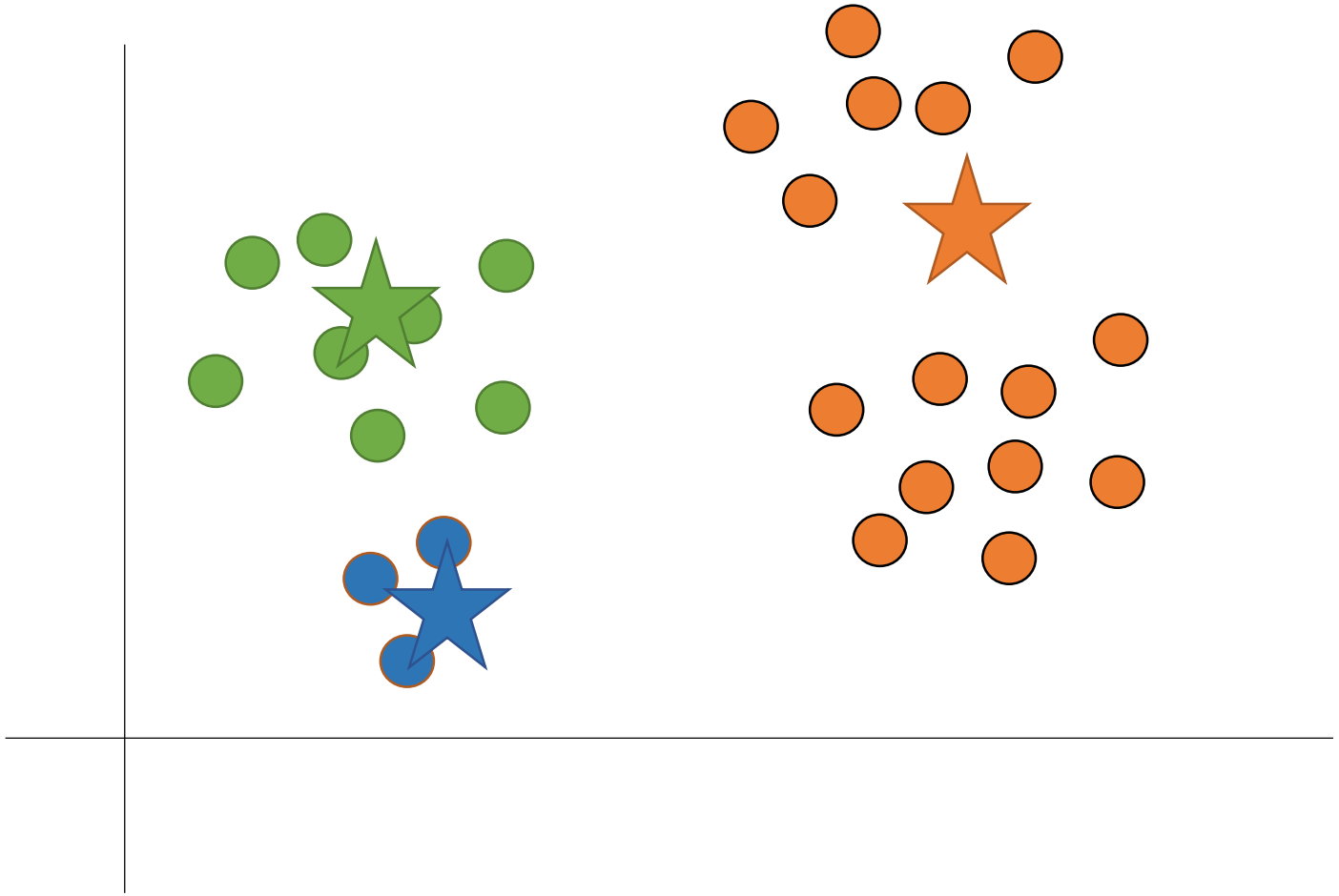
How to choose *k*



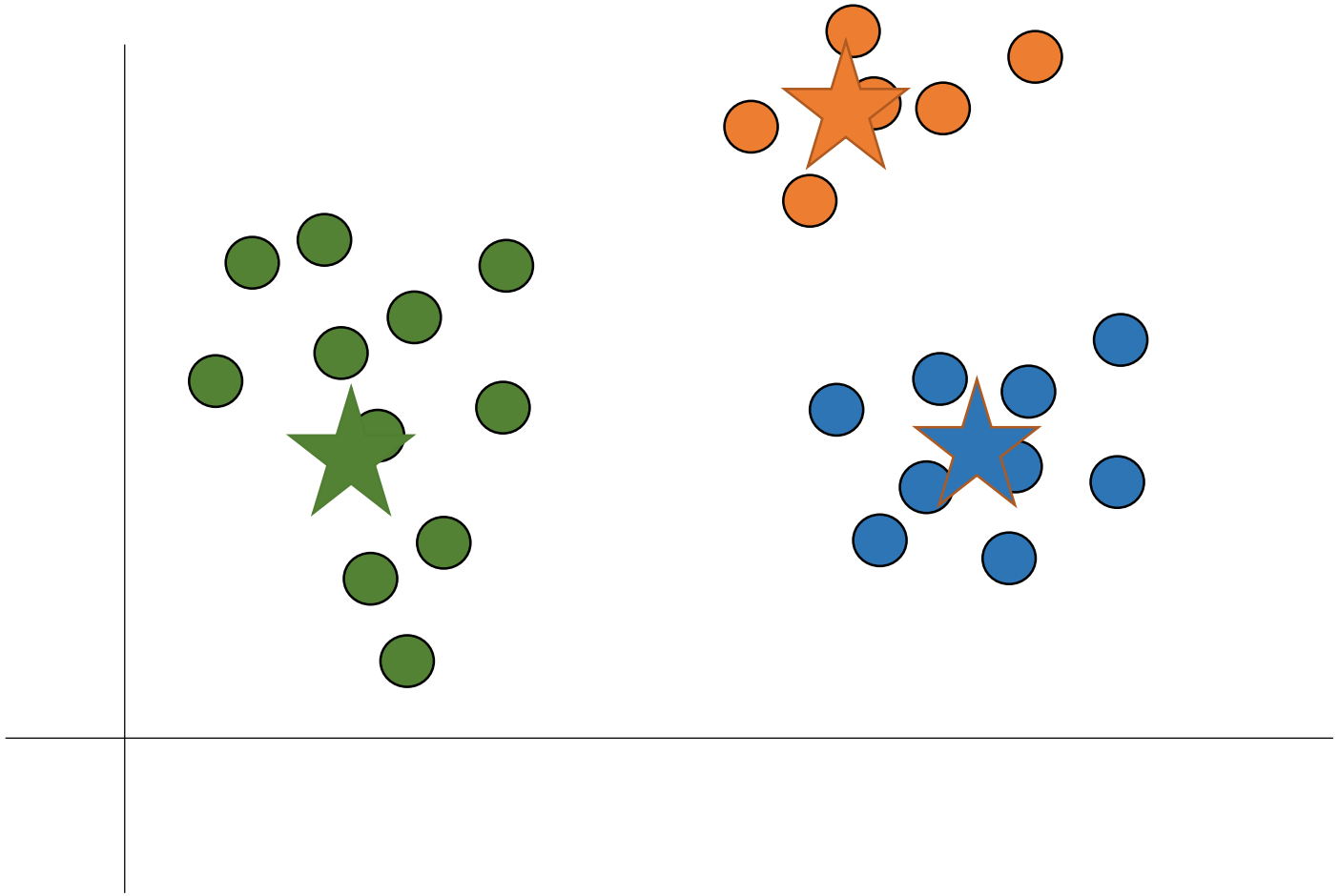
Unlucky Initialization



Unlucky Initialization



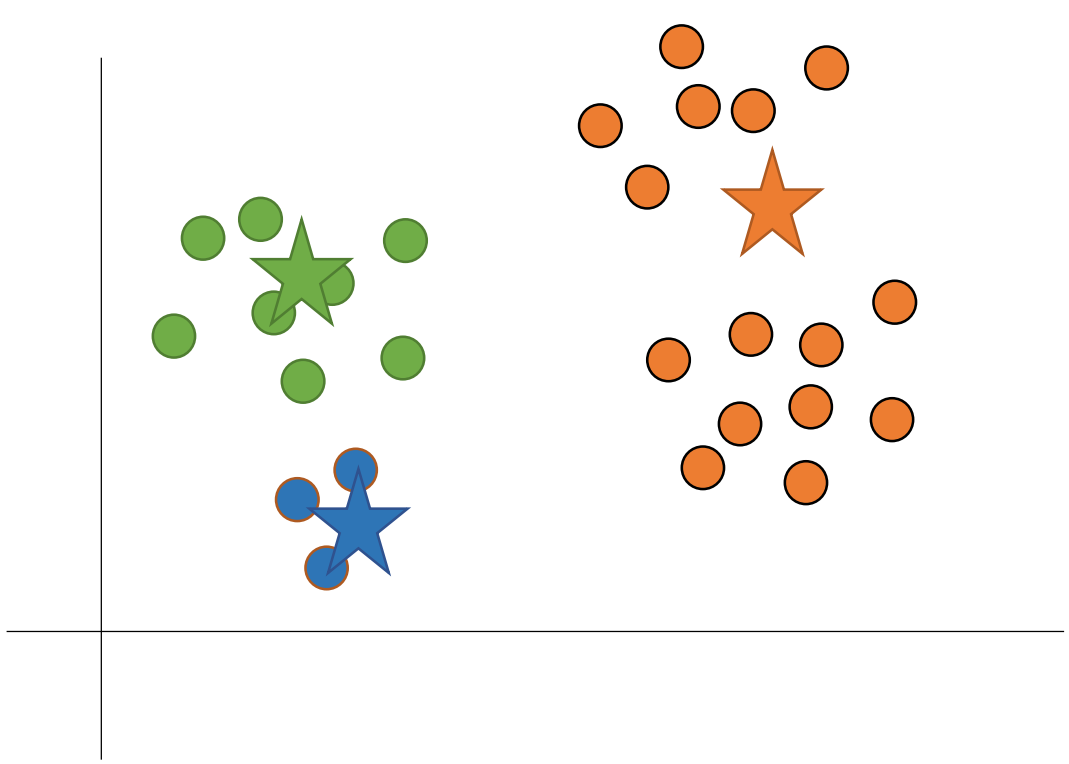
What we wanted…

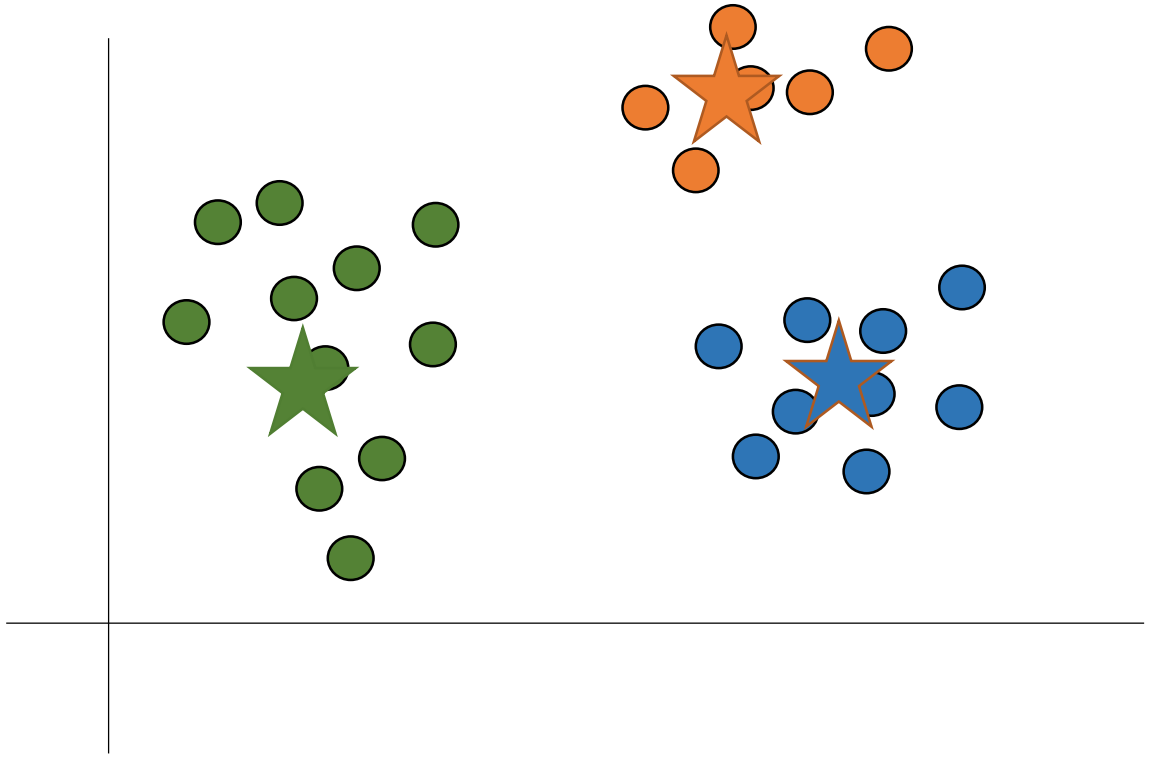


Solutions

- Run k-means multiple times
- Select the one with least $dissimilarity(C)$

Solutions





$$dissimilarity(C) \geq dissimilarity(C)$$