# Lecture: K-Means Clustering

Rodrigo Canaan
rodrigocanaan.com
rocanaan@gmail.com
@rocanaan

# Slides and Code

https://github.com/rocanaan/k-means

# Types of Machine Learning

- **Supervised Learning**
  Learning to predict values or classify objects based on labeled data

- **Unsupervised Learning**
  Learning patterns based on unlabeled data

- **Reinforcement Learning**
  Learning to act intelligently based on interaction with an environment

# Types of Machine Learning

- **Supervised Learning**

Learning to predict values or classify objects based on labeled data

- **Unsupervised Learning**

Learning patterns based on unlabeled data

- **Reinforcement Learning**

Learning to act intelligently based on interaction with an environment

# Why learn from unlabeled data?

- It's *everywhere*!
- Human labeling is expensive!

# What can we do with unlabeled data?

- Gain insight about the data
- Compress and/or visualize the data
- Generate examples that look like the data
- Label data for downstream tasks

# Examples in various domains

- **Visual Processing**
Group similar images even without labels

- **Recommender Systems**
Group similar products and/or users together

- **Games**
Build models of different player styles

- **Science**
Create a taxonomy of phenomena (e.g. stars) based on their observed properties (magnitude, spectrum, distance…)

# Unsupervised Learning Tasks

- **Clustering**

Partitioning the data into “clusters” based on a measure of similarity

- **Dimensionality reduction**

Transforming the data from a high dimension to a low dimension

- **Generative models**

Learning a “model” that can be used to produce new examples that are similar to the data

# Unsupervised Learning Tasks

- **Clustering**

  Partitioning the data into “clusters” based on a measure of similarity

- **Dimensionality reduction**

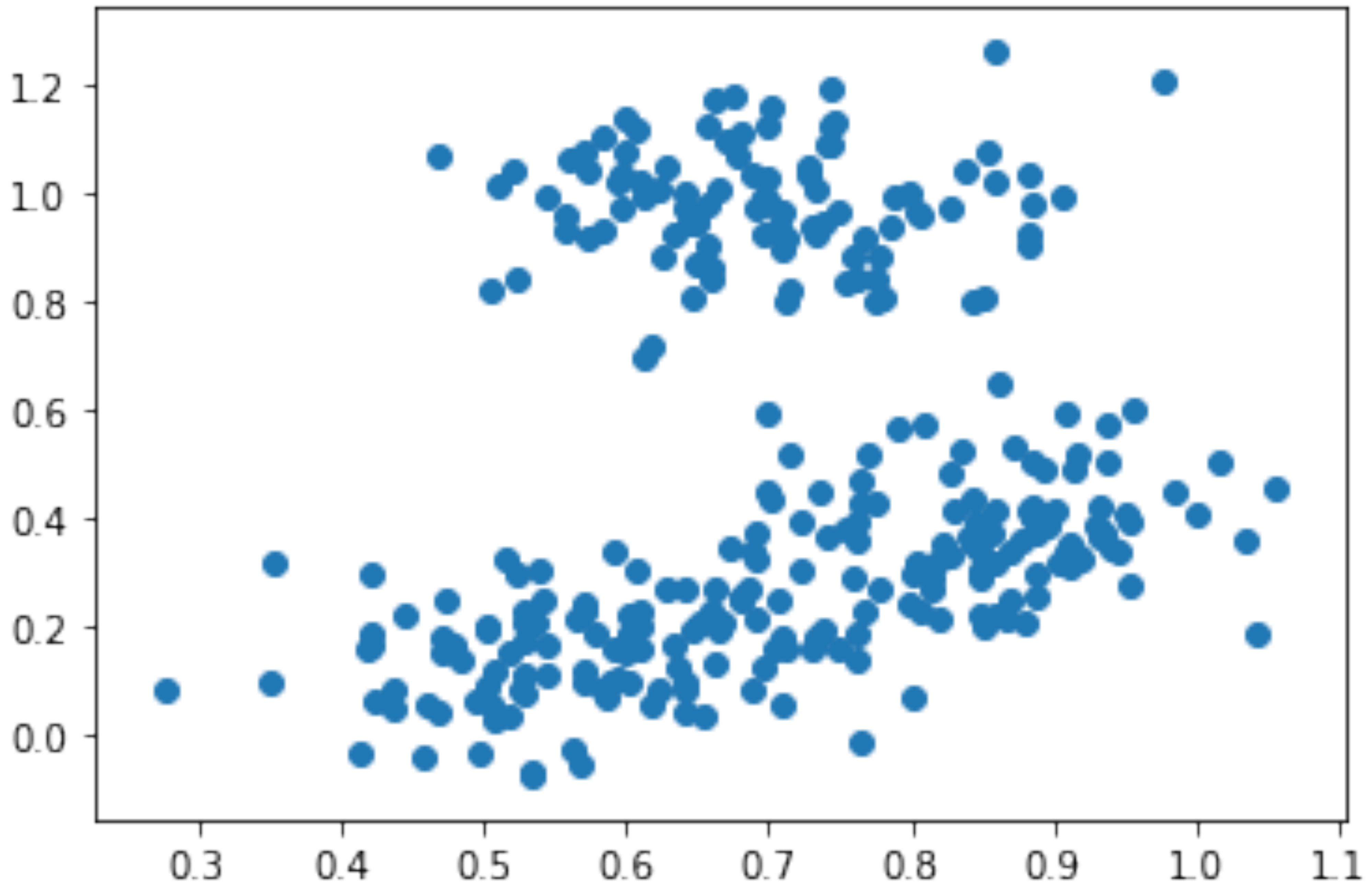
  Transforming the data from a high dimension to a low dimension

- **Generative models**

  Learning a “model” that can be used to produce new examples that are similar to the data

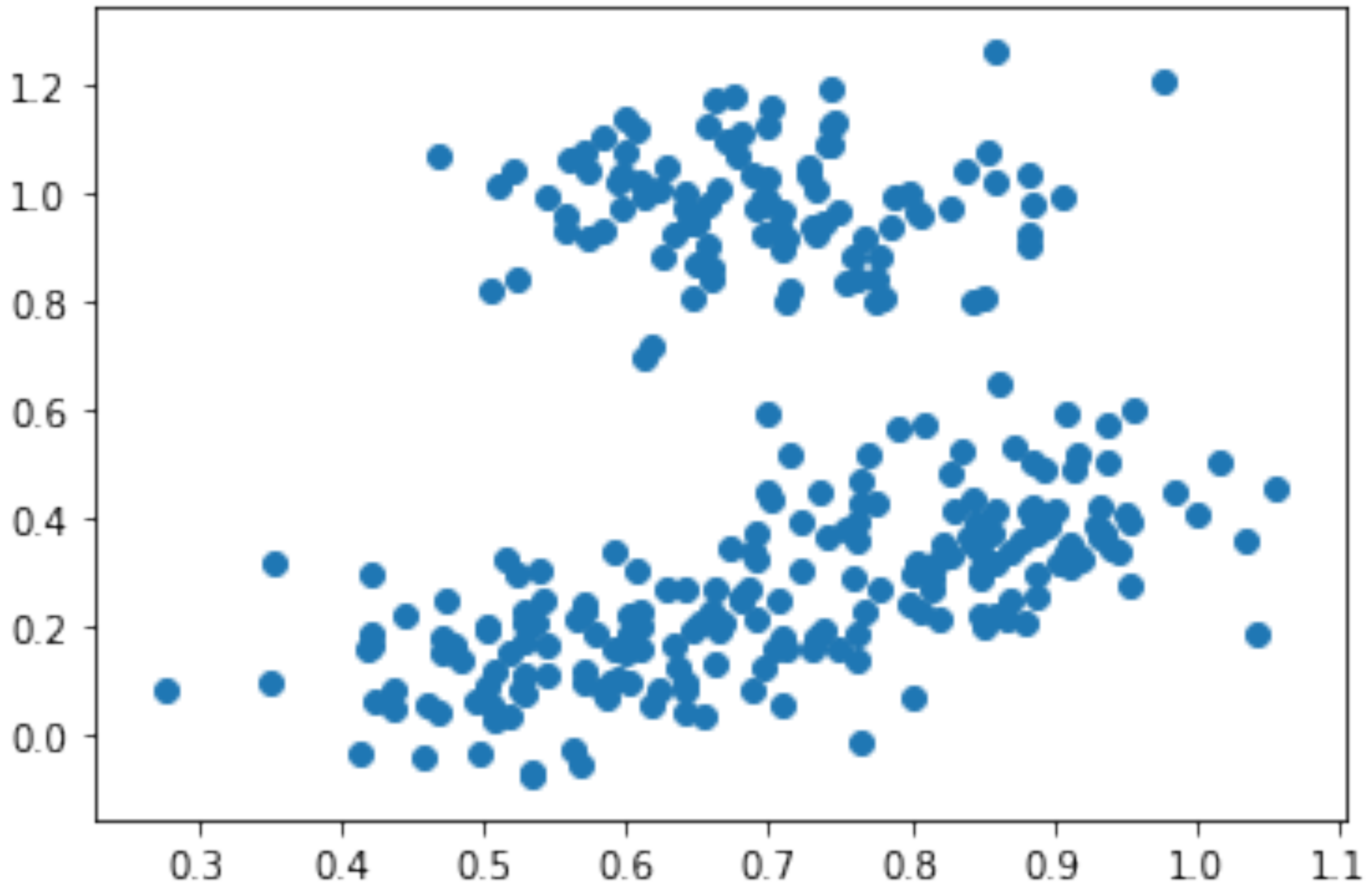
# Clustering - Visual Example

**Before clustering**

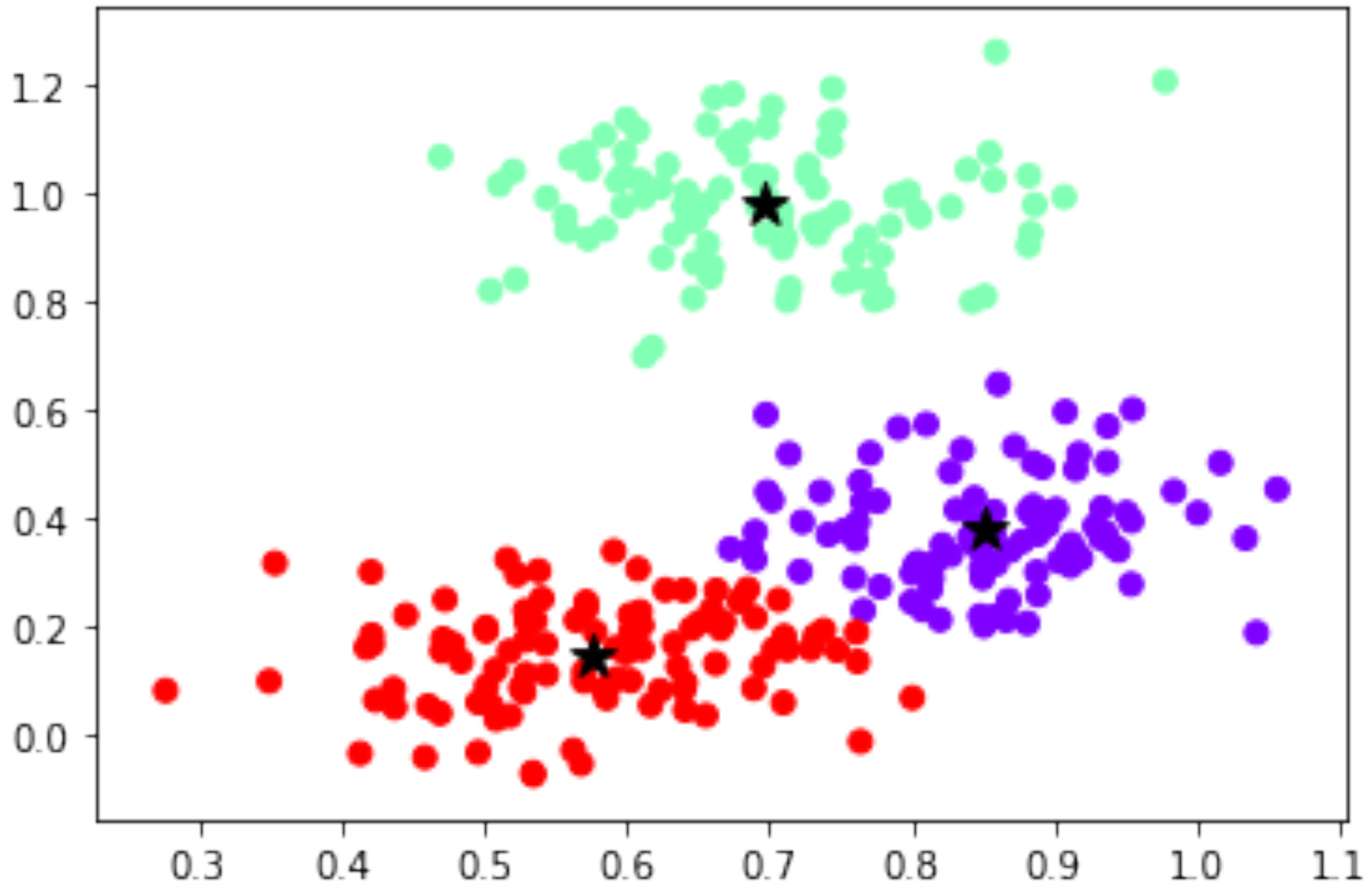


# Clustering - Visual Example

**Before clustering**




**After k-means Clustering (k=3)**



# Clustering - Distances

- **What does it mean for two data points to be "similar"?**

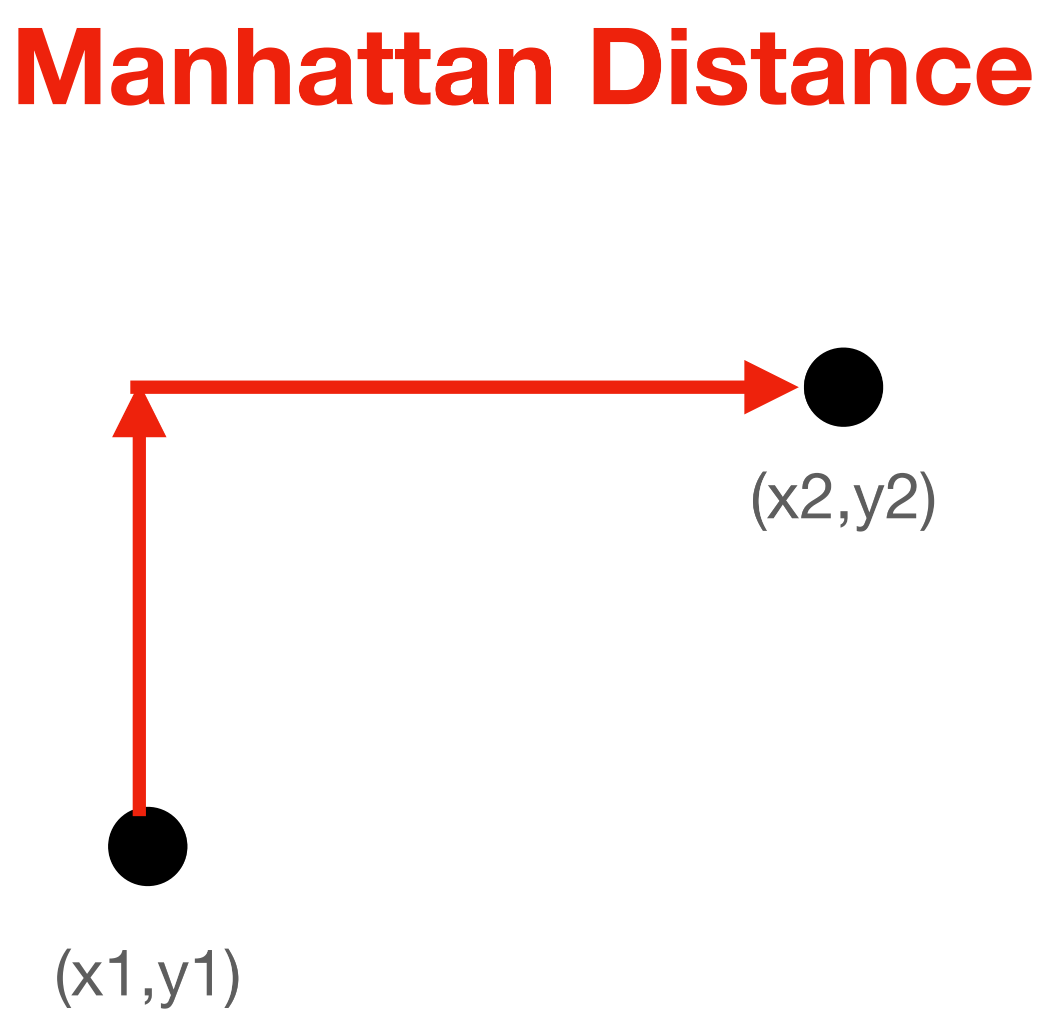
Usually, it means to have a small "distance" according to some metric

# Clustering - Distances

- **What does it mean for two data points to be "similar"?**

Usually, it means to have a small "distance" according to some metric

**Examples** (in 2-dimensions)




$$D_M = |x2 - x1| + |y2 - y1|$$

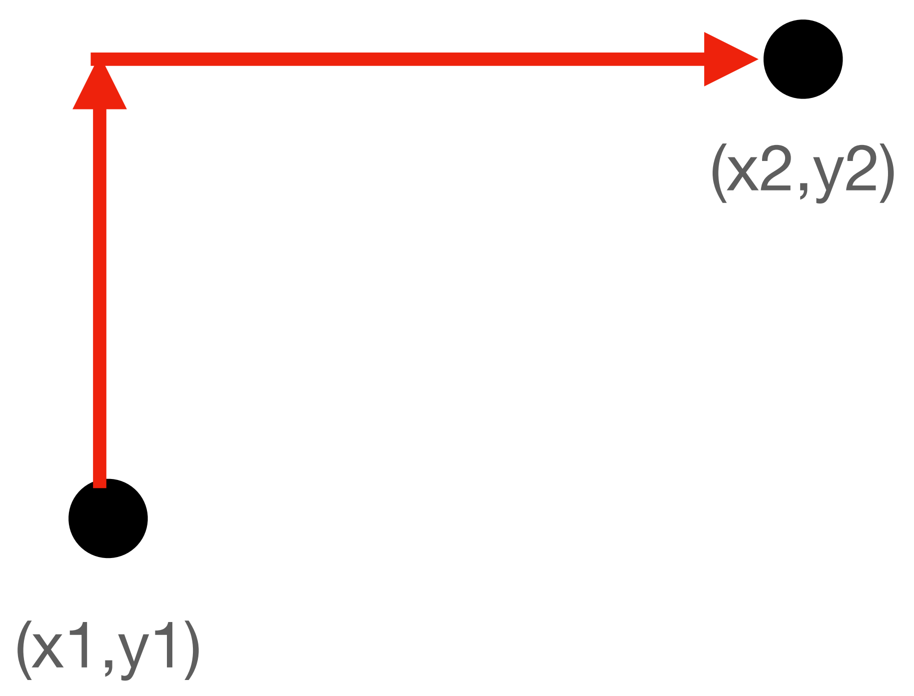
# Clustering - Distances

- **What does it mean for two data points to be "similar"?**

Usually, it means to have a small "distance" according to some metric

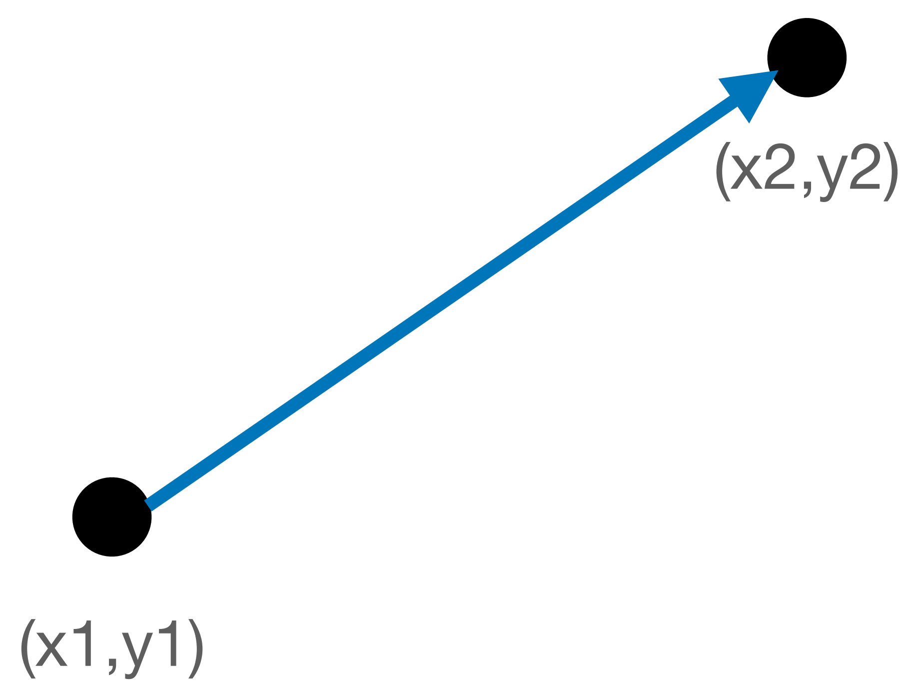
**Examples** (in 2-dimensions)

**Manhattan Distance**




$$D_M = |x2 - x1| + |y2 - y1|$$

Euclidean Distance




$$D_E = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

# Clustering Objectives

## What does it mean for a clustering to be “good”?

Intuitively:

- Elements in the same cluster are similar (“close”)
- Elements in different clusters are dissimilar (“far”)

**Examples:**

- Within-Cluster Sum of Squares (WCSS)
- Average Distance from Centroid
- Maximum Distance from Centroid

What is the simplest-possible clustering algorithm?

# Clustering by exhaustive enumeration

1. Enumerate all possible clusterings

2. Evaluate clustering objective for each clustering

3. Return the best clustering

# Clustering by exhaustive enumeration

1. Enumerate all $k^n$ possible clusterings
2. Evaluate clustering objective for each clustering
3. Return the best clustering

# Clustering by exhaustive enumeration

1. Enumerate all k^n possible clusterings

2. Evaluate clustering objective for each clustering

3. Return the best clustering

**Very expensive!**

Need **heuristic** for approximate (but faster) solution!

# Clustering by exhaustive enumeration

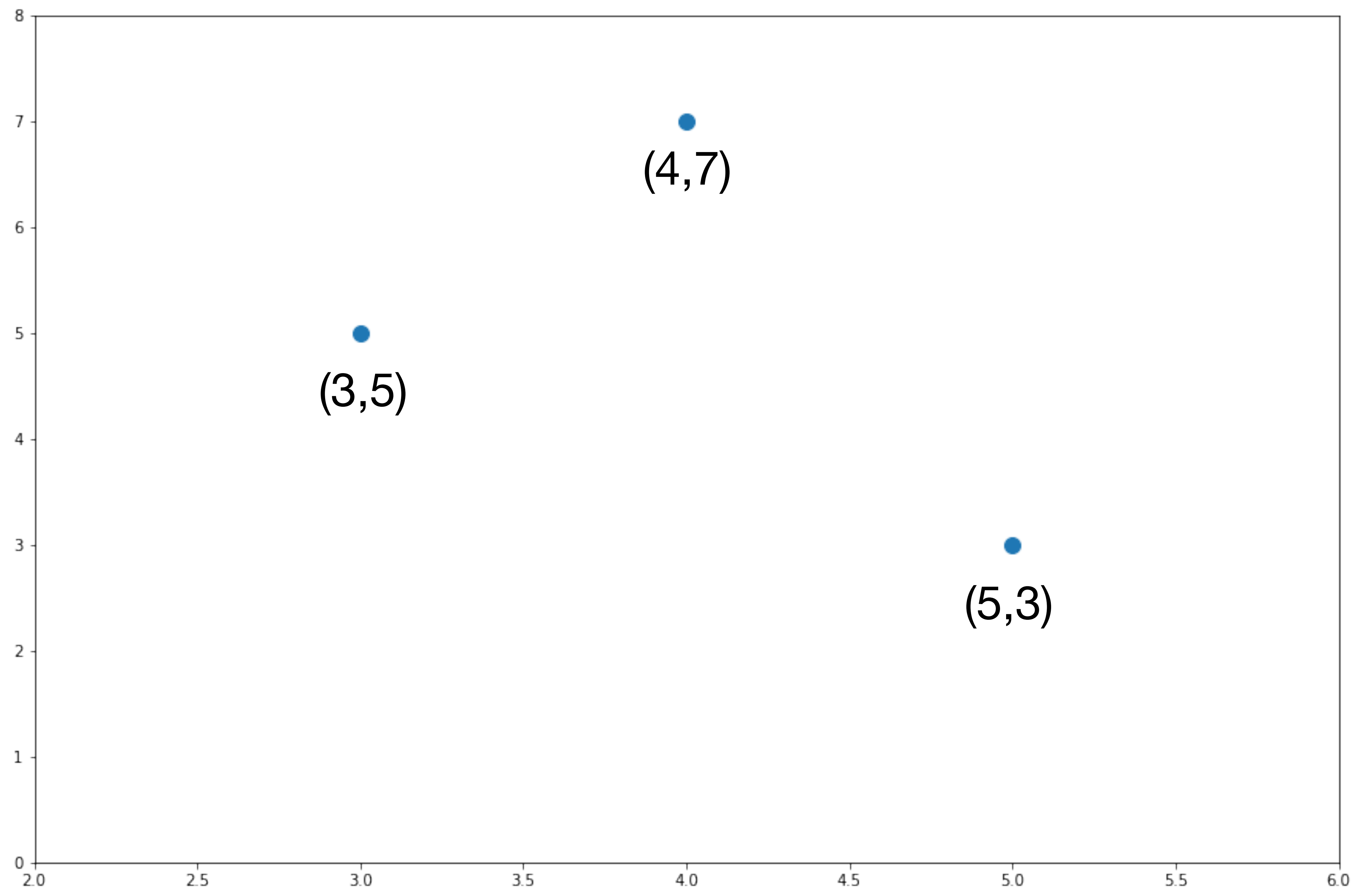
1. Enumerate all k^n possible clusterings

2. Evaluate clustering objective for each clustering

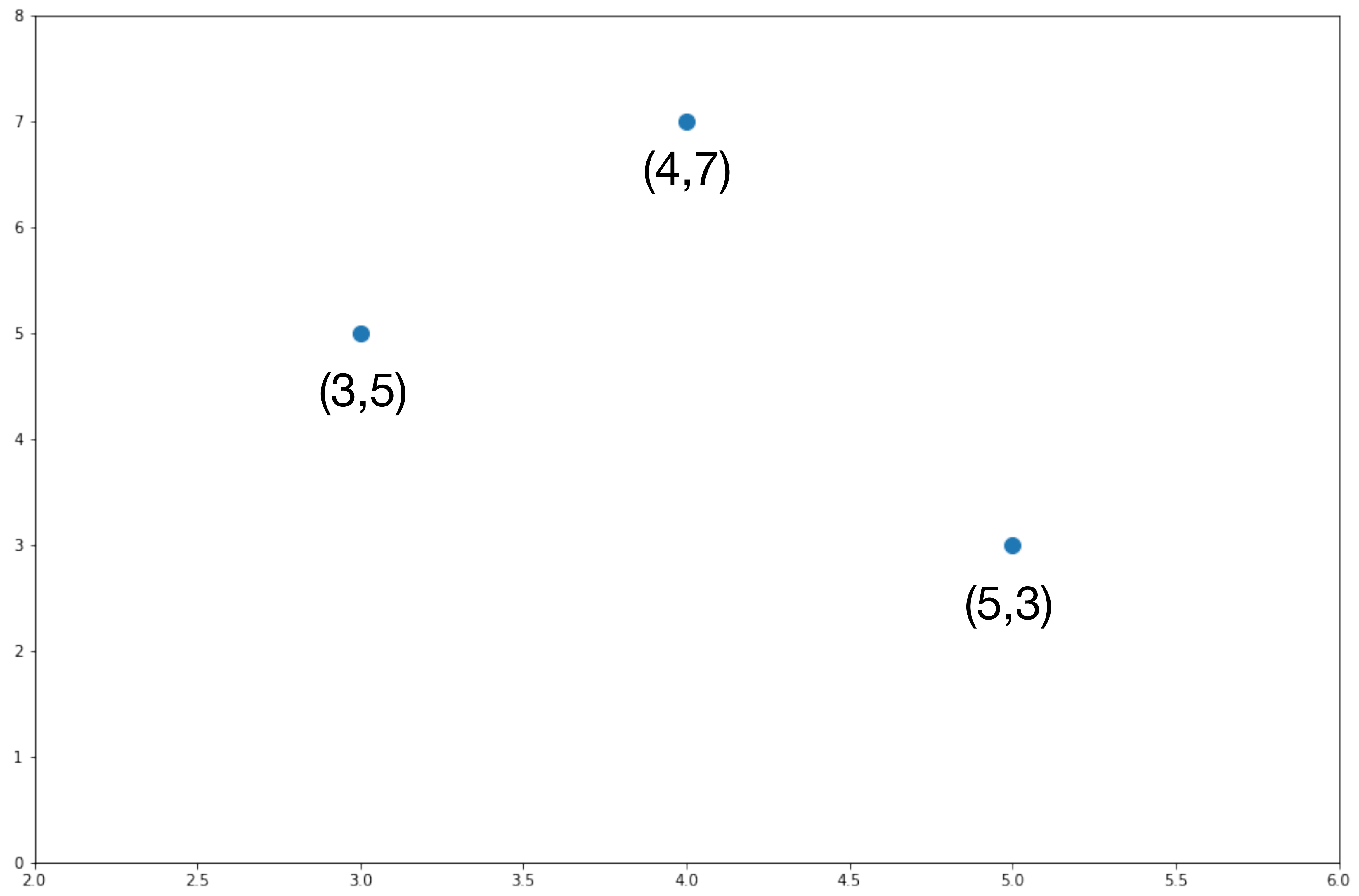
3. Return the best clustering

**Very expensive!**
Need **heuristic** for approximate (but faster) solution!
**Example:** assign each point to cluster with nearest centroid

# Centroid Calculation

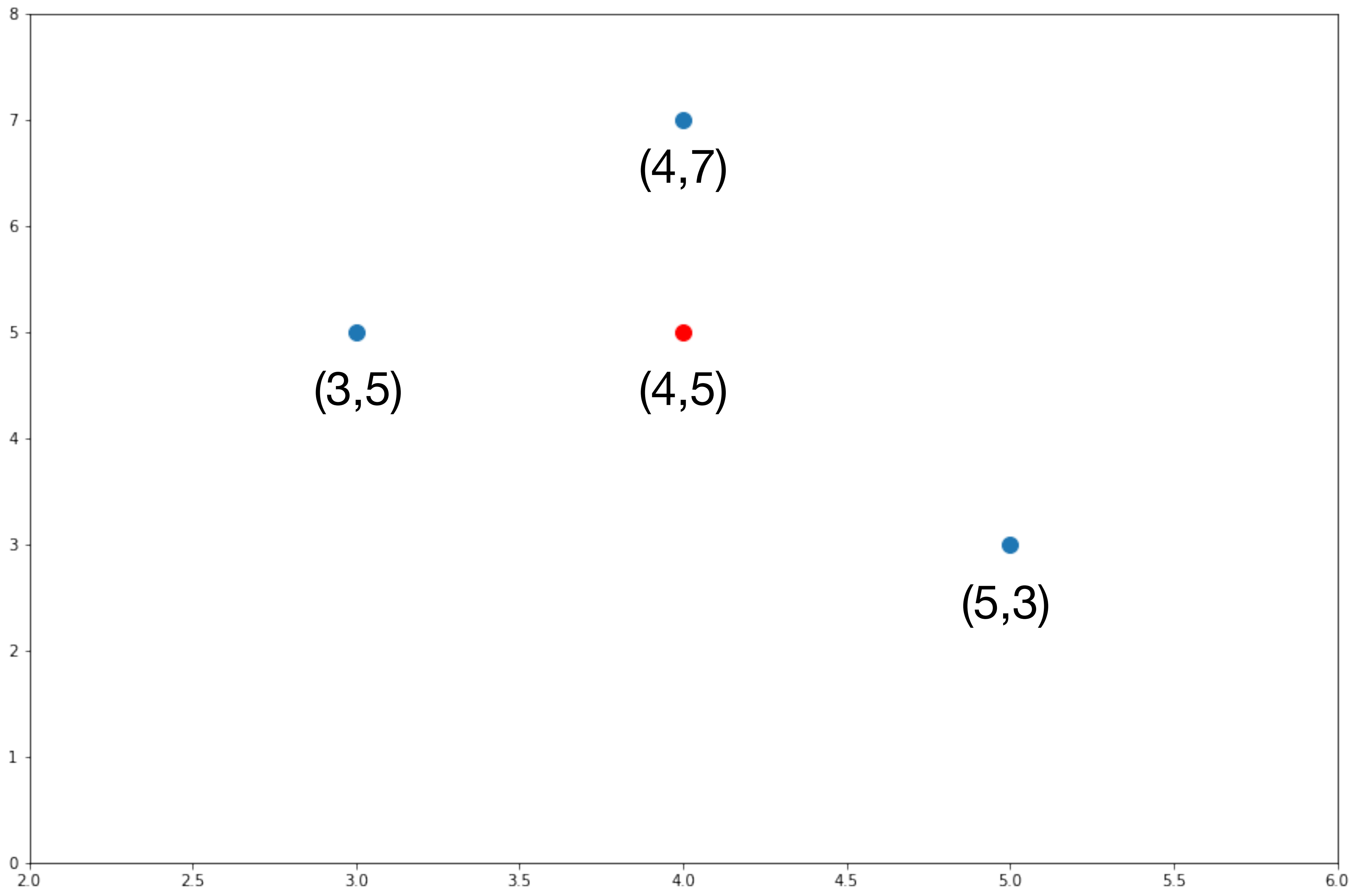


# Centroid Calculation




x_mean = (3+4+5)/3 = 4
y_mean = (5+7+3)/3 = 5

# Centroid Calculation




x_mean = (3+4+5)/3 = 4
y_mean = (5+7+3)/3 = 5

Centroid = (4,5)

# Outline of K-Means Clustering Algorithm

1. **Generate *k* initial centroids**
2. **Assign points based on heuristic**
   Each point is assigned to cluster with nearest centroid by Euclidean distance
3. **Recalculate centroids**
4. **Repeat (2, 3) until no re-assignments**
   Alternatively, up to some maximum number of epochs

# (Naive) K-Means Pseudo-Code

```
# Pseudo-code of K-Means clustering algorithm
# Assumes a Partition class with methods to maintain centroids and labels
# P and P_new are instances of this Partition class
Function k-means (data,k)
    P <- initialize_partition(data,k)
    stop <- False
    while not stop # runs until no new assignments
                   # optionally, until some max_iterations is reached
        P_new <- empty_partition(k)
        for d in data # Check distance from d to k centroids, assign to closest
            new_label <- P.get_closest_centroid_label(d)
            P_new.add_element(d,new_label)
        Endfor
        if P_new = P # If nothing changed, stop
            stop = True
        Endif
        P <- P_new
        P.compute_centroids() # Re-compute centroids based on new labels
    Endwhile

    return P
Endfunction

```

# (Naive) K-Means Pseudo-Code

```
# Pseudo-code of K-Means clustering algorithm
# Assumes a Partition class with methods to maintain centroids and labels
# P and P_new are instances of this Partition class
function k-means (data,k)
    P <- initialize_partition(data,k)
    stop <- False
    while not stop # runs until no new assignments
                   # optionally, until some max_iterations is reached
        P_new <- empty_partition(k)
        for d in data # Check distance from d to k centroids, assign to closest
            new_label <- P.get_closest_centroid_label(d)
            P_new.add_element(d,new_label)
        Endfor
        if P_new = P # If nothing changed, stop
            stop = True
        Endif
        P <- P_new
        P.compute_centroids() # Re-compute centroids based on new labels
    Endwhile

    return P
Endfunction
```

# K-Means Initialization

## How to initialize clusters?

- **Forgy method:** Choose *k* initial centroids randomly (from the data), assign other points according to distance to centroids

- **Random Partition:** Assign each datapoint to a random cluster label, then compute centroids

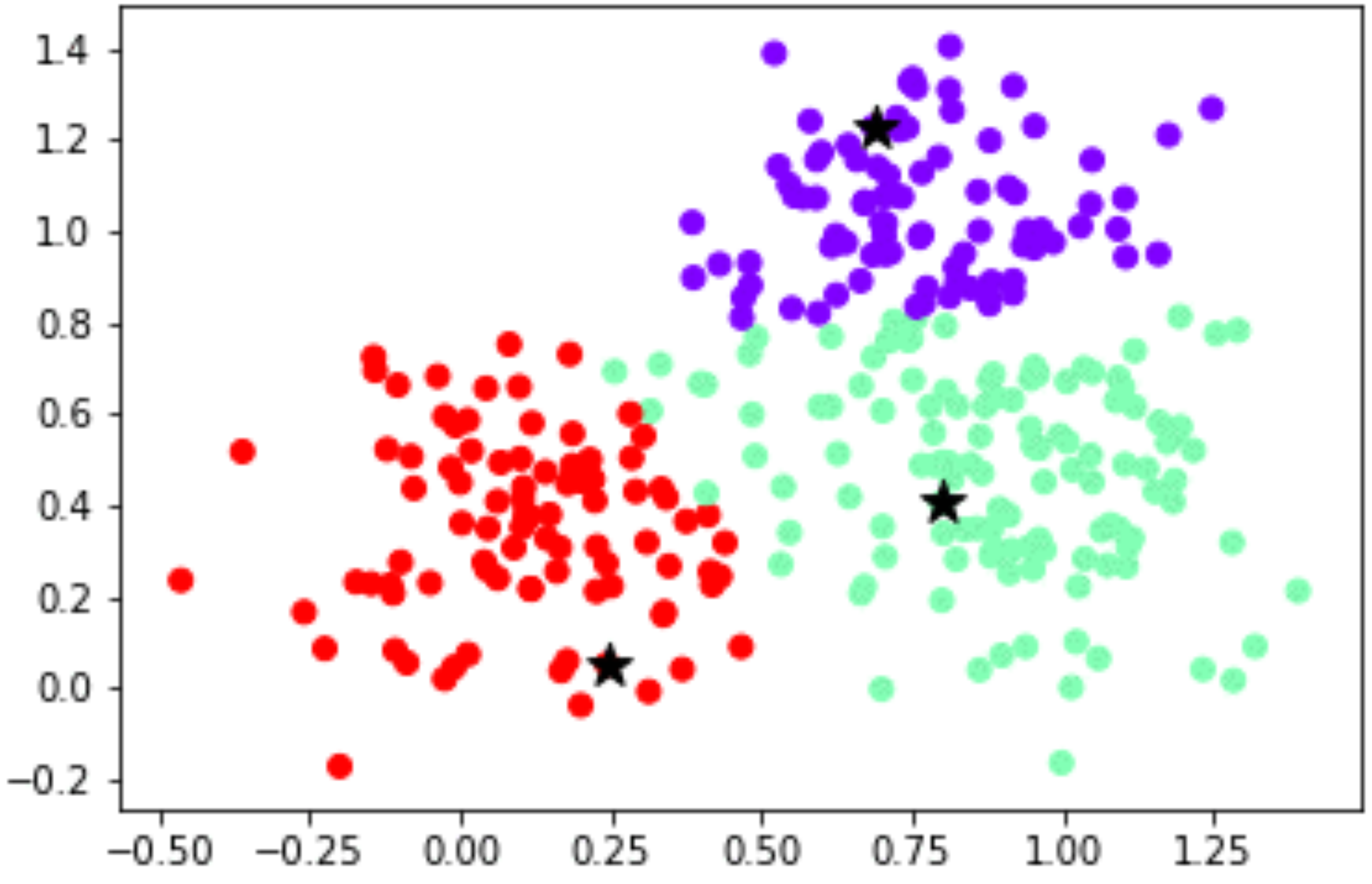
- Other options: see comparative study by (Celebi et al., 2013)

# K-Means Initialization Pseudo-Code

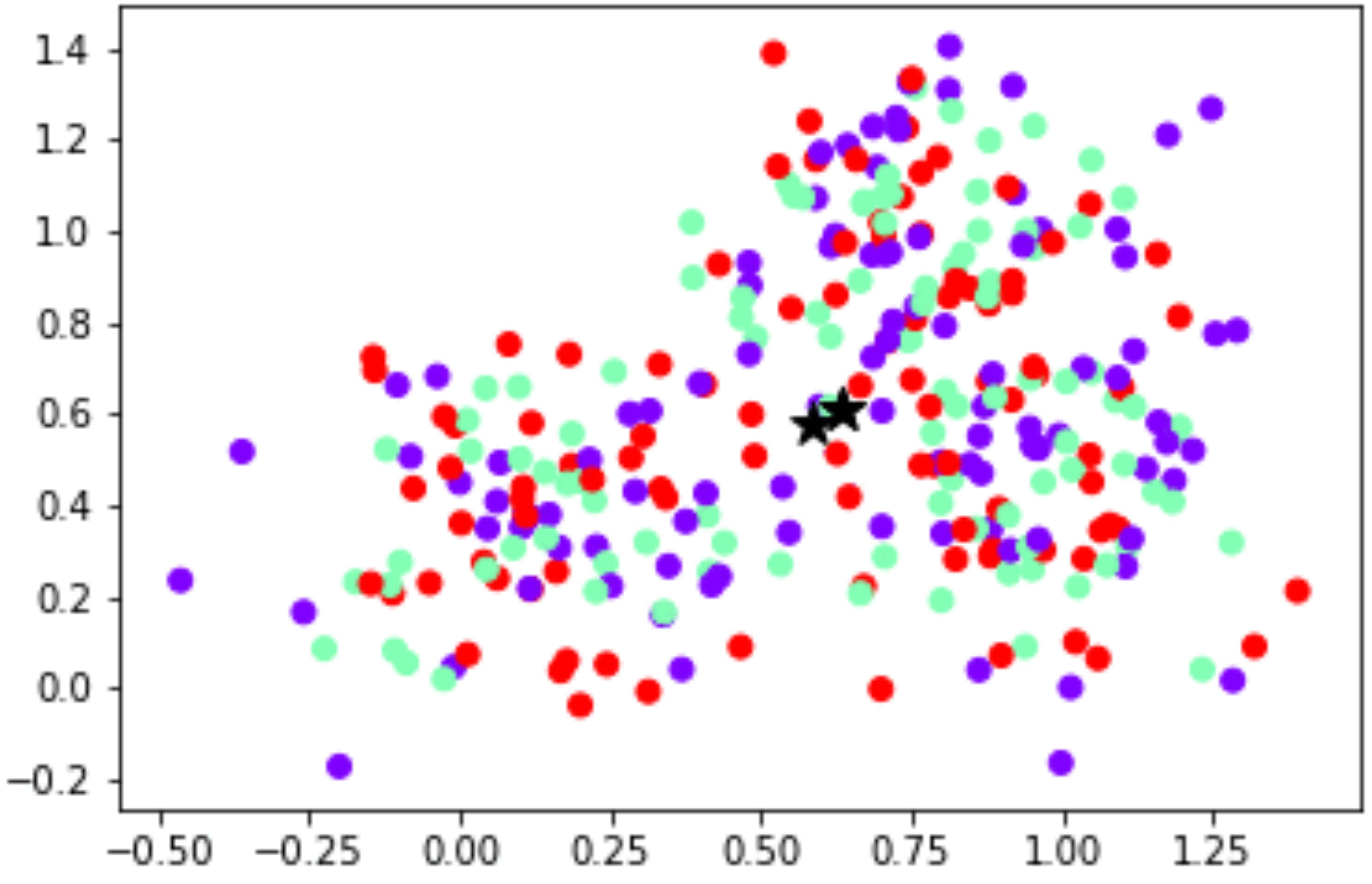
```
# Initial assignment of labels can be done in the main algorithm itself!
Function initialize_partitions(data,k)
    P <- empty_partition(k)
    P.centroids <- sample_without_replacement(data,k) # get k points from data
Endfunction

#OR

# Random Partition initialization
# Assign each point to a random label, then compute centroids
Function initialize_partitions(data,k)
    P <- empty_partition(k)
    for d in data
        label <- random_uniform(k)
        P.add_element(d,label)
    Endfor
    P.compute_centroids()
    return P
Endfunction
```

# Demo

**Forgy Method, k=3**




**Random Partition, k=3**



# Advantages and Disadvantages

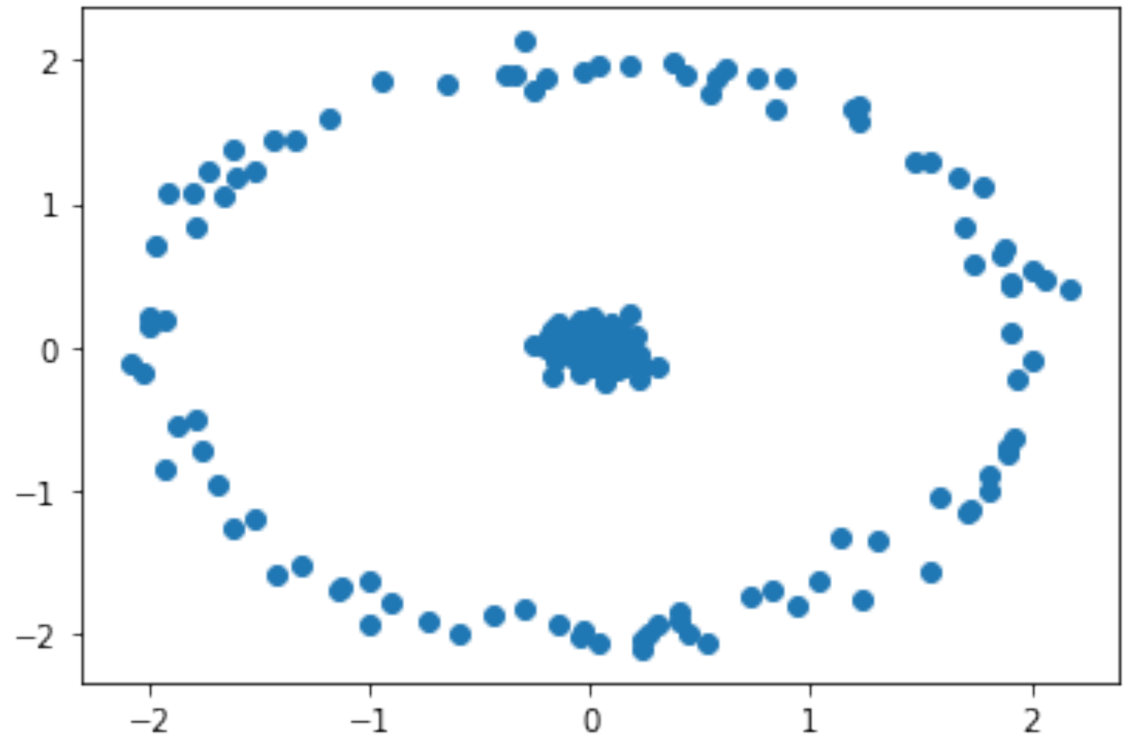
# (Naive) K-Means Disadvantages

- Struggles with some data shapes, sizes and outliers

# (Naive) K-Means Disadvantages

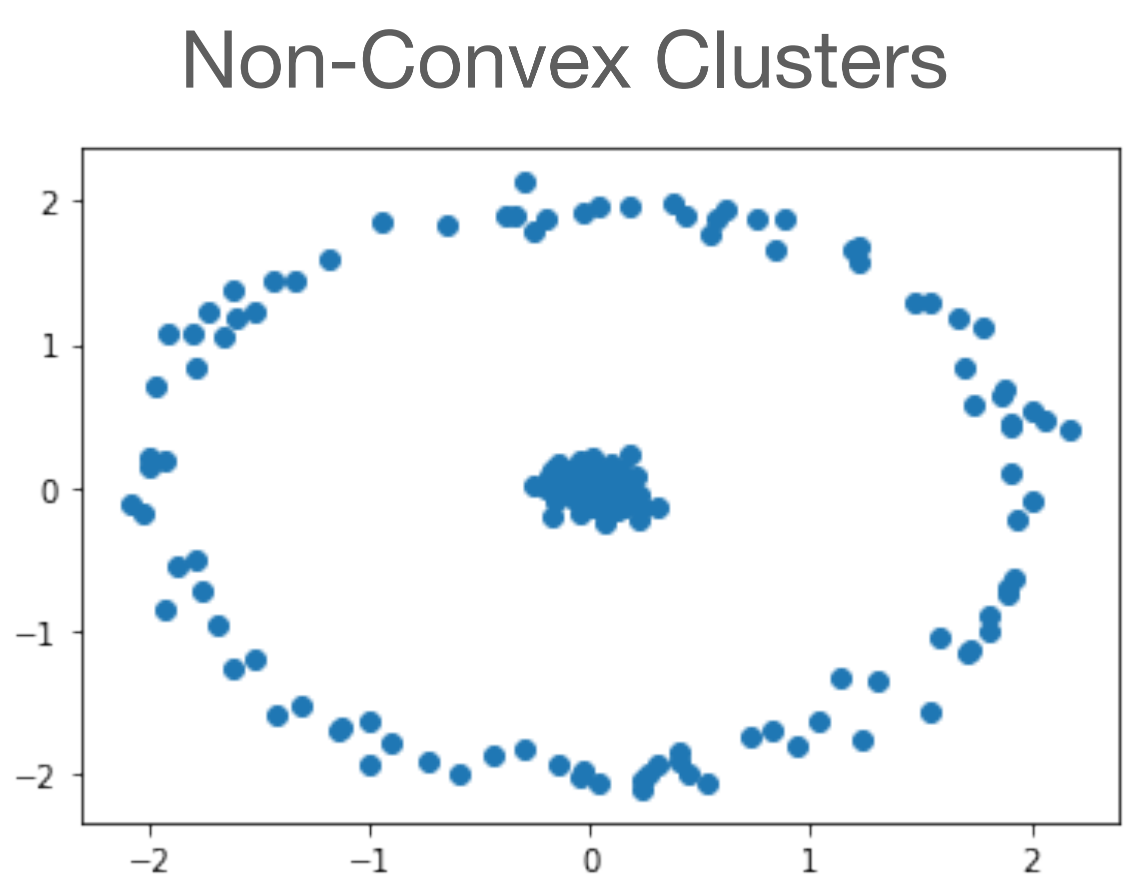
- **Struggles with some data shapes, sizes and outliers**

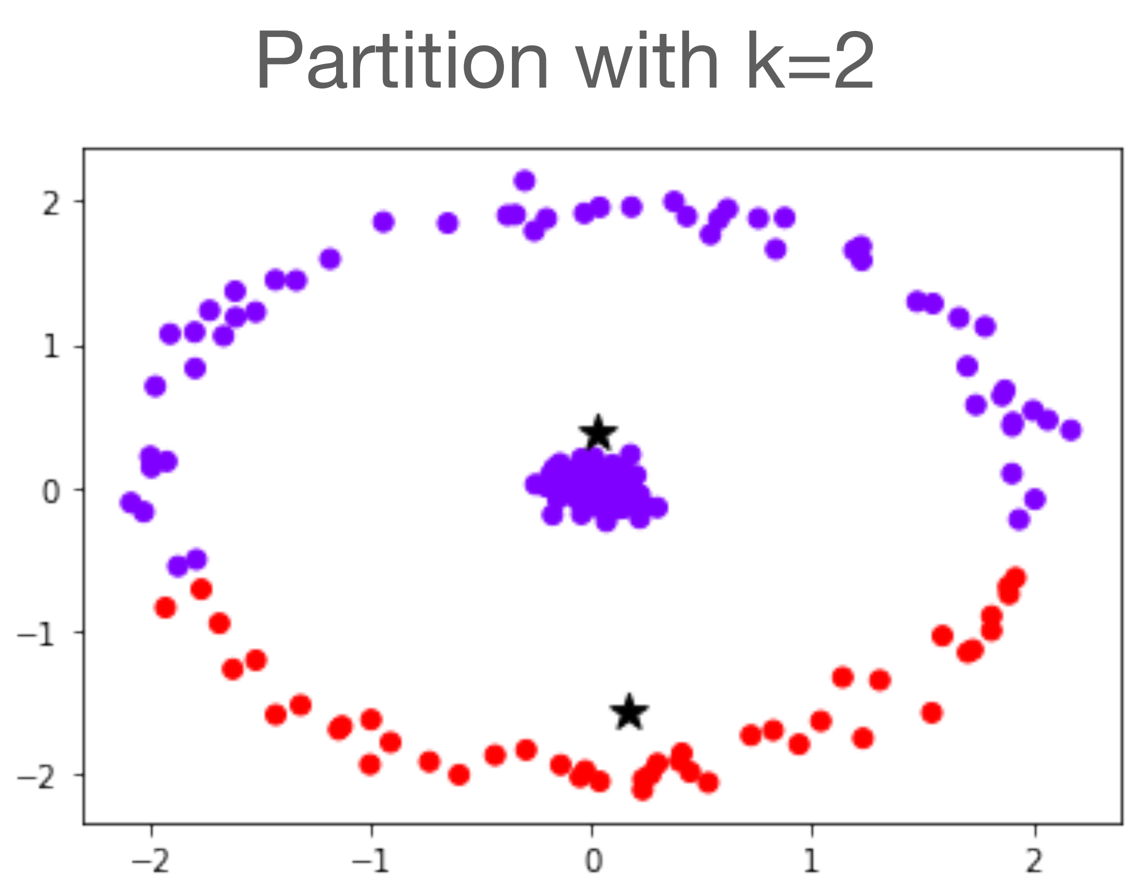
Non-Convex Clusters



# (Naive) K-Means Disadvantages

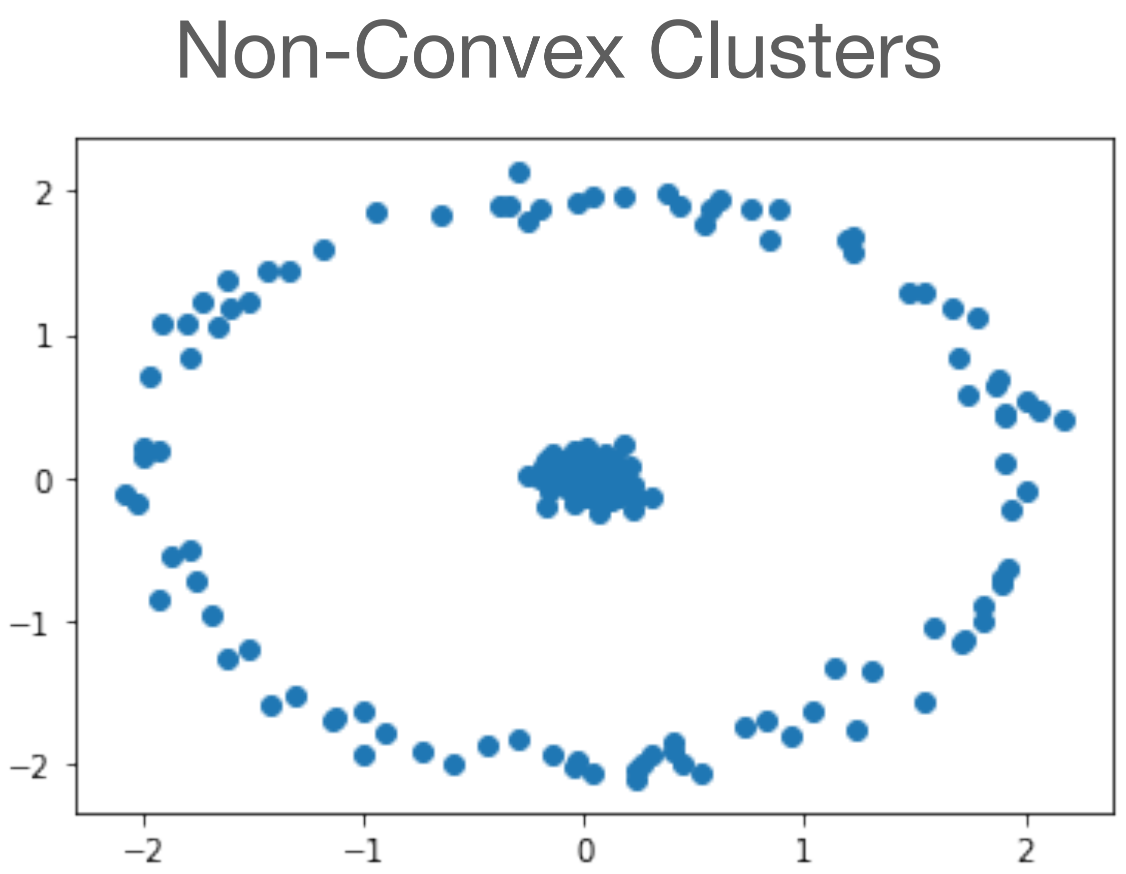
- **Struggles with some data shapes, sizes and outliers**

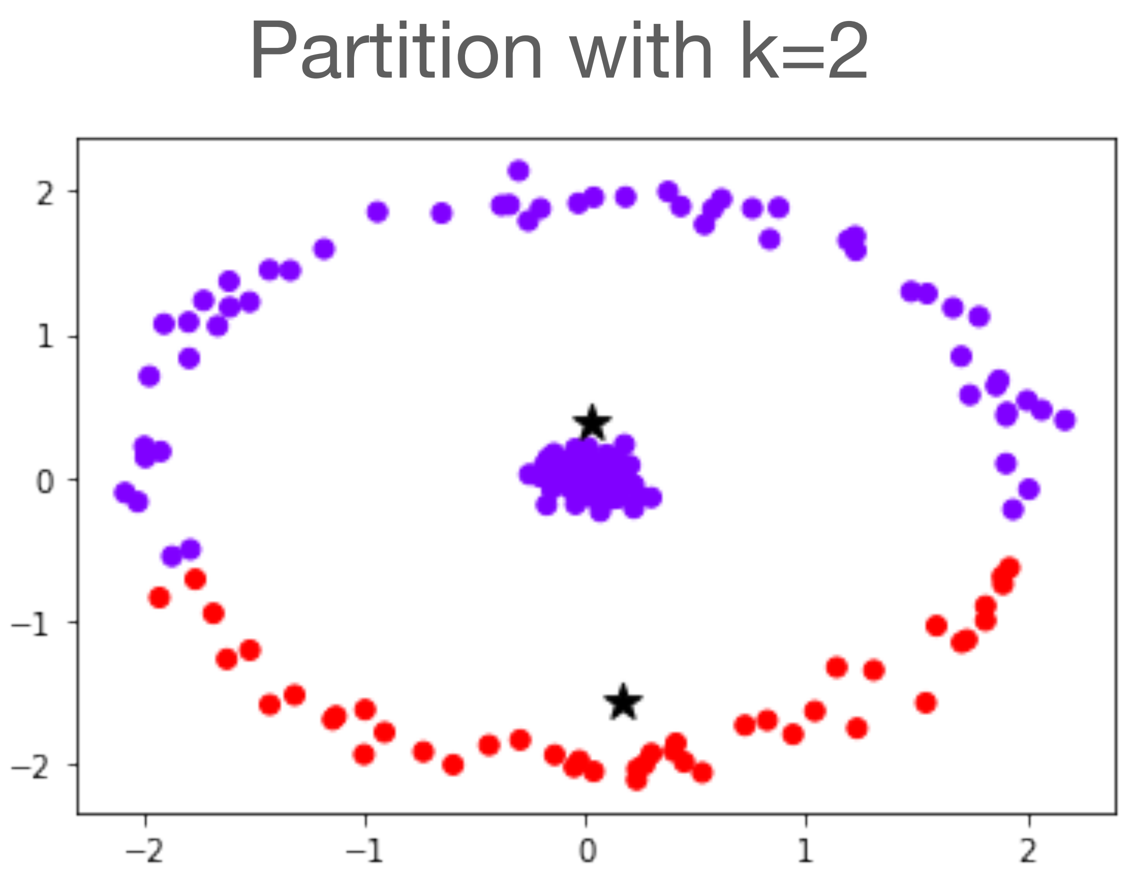


# (Naive) K-Means Disadvantages

- **Struggles with some data shapes, sizes and outliers**

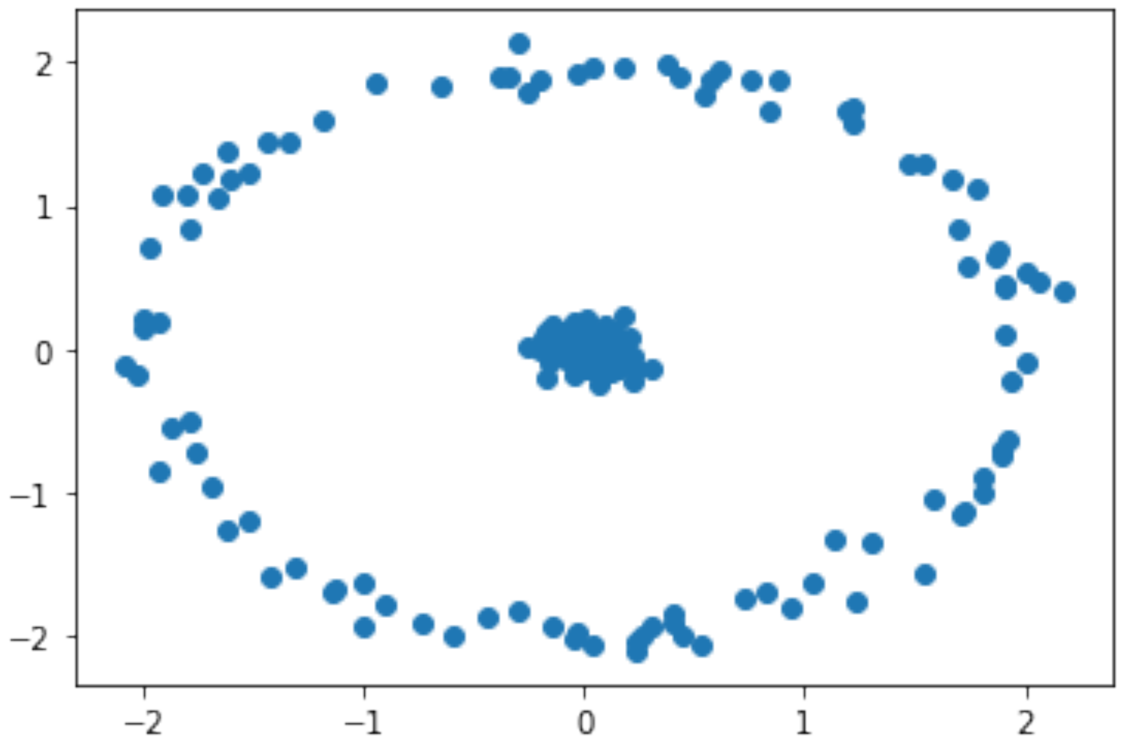


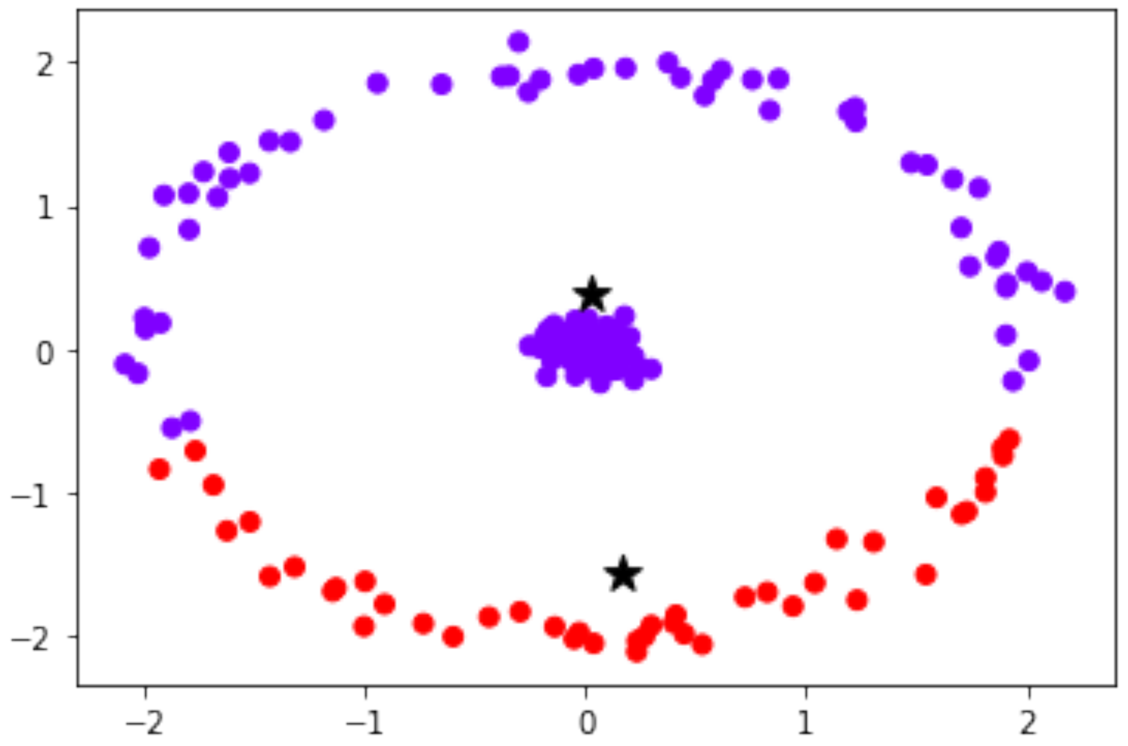

# (Naive) K-Means Disadvantages

- **Struggles with some data shapes, sizes and outliers**

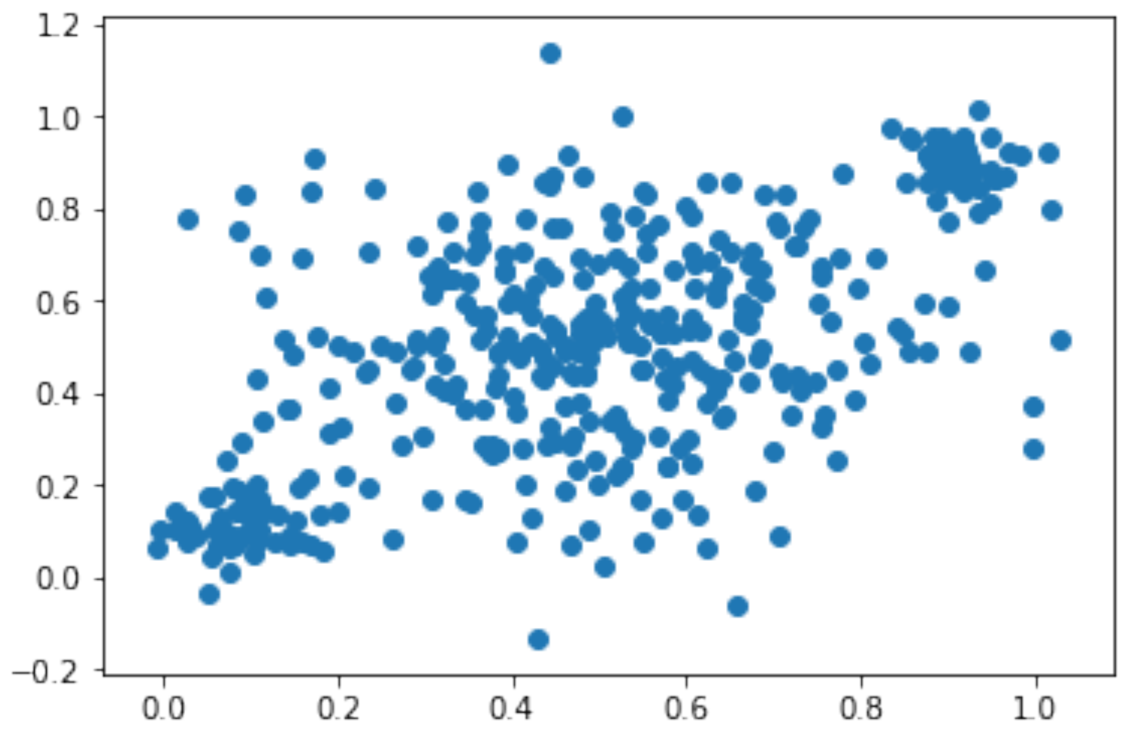
Non-Convex Clusters



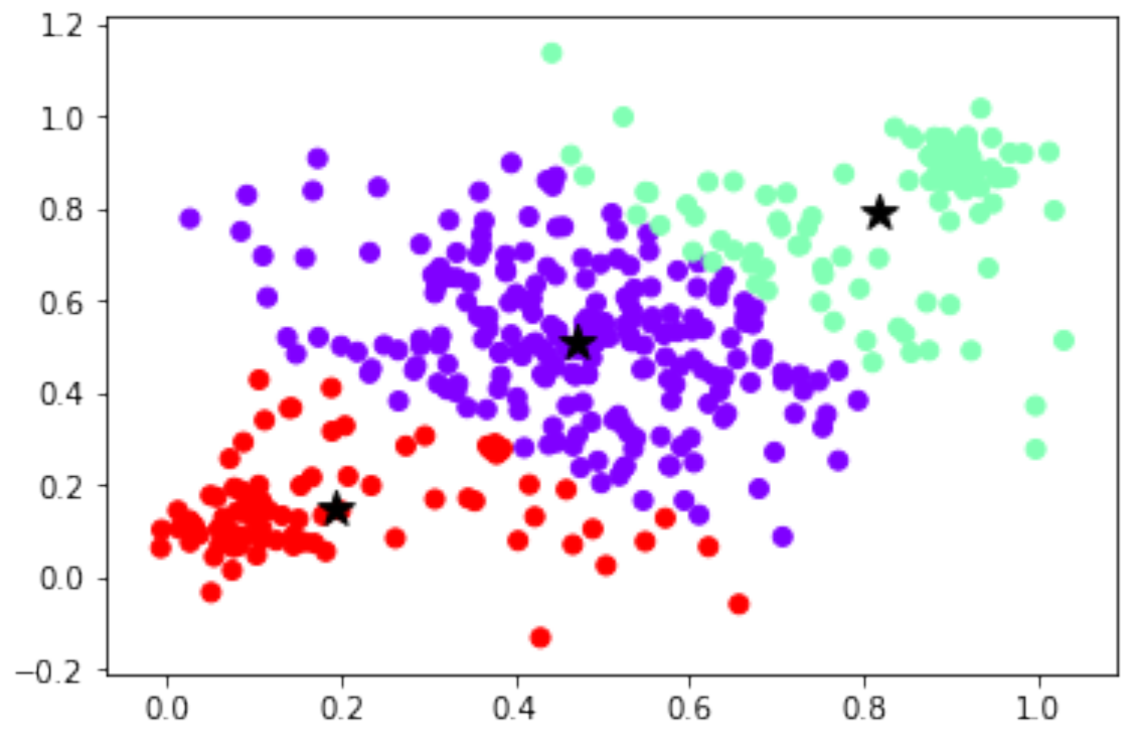

Partition with k=2




Clusters of different sizes




Partition with k=3



# (Naive) K-Means Disadvantages

- Struggles with some data shapes, sizes and outliers
- Struggles with categorical data
- Guaranteed convergence only to local (not global optimum)
- Need to specify *k* in advance
- “Curse of dimensionality”
- Worst-case time complexity?

# Time Complexity

O(e)
Outer-loop epochs

O(n)
Inner-loop iterations

O(k)

```
# Pseudo-code of K-Means clustering algorithm
# Assumes a Partition class with methods to maintain centroids and labels
# P and P_new are instances of this Partition class
Function k-means (data,k)
    P <- initialize_partition(data,k)
    stop <- False
    while not stop # runs until no new assignments
                   # optionally, until some max_iterations is reached
        P_new <- empty_partition(k)
        for d in data # Check distance from d to k centroids, assign to closest
            new_label <- P.get_closest_centroid_label(d)
            P_new.add_element(d,new_label)
        Endfor
        if P_new = P # If nothing changed, stop
            stop = True
        Endif
        P <- P_new
        P.compute_centroids() # Re-compute centroids based on new labels
    Endwhile

    return P
Endfunction
```

# Time Complexity = O(nke)

O(e)
Outer-loop
epochs

O(n)
Inner-loop
iterations

```
# Pseudo-code of K-Means clustering algorithm
# Assumes a Partition class with methods to maintain centroids and labels
# P and P_new are instances of this Partition class
Function k-means (data,k)
    P <- initialize_partition(data,k)
    stop <- False
    while not stop # runs until no new assignments
                   # optionally, until some max_iterations is reached
        P_new <- empty_partition(k)
        for d in data # Check distance from d to k centroids, assign to closest
            new_label <- P.get_closest_centroid_label(d)
            P_new.add_element(d,new_label)
        Endfor
        if P_new = P # If nothing changed, stop
            stop = True
        Endif
        P <- P_new
        P.compute_centroids() # Re-compute centroids based on new labels
    Endwhile

    return P
Endfunction
```

O(k)

**Note: number of epochs e is hard to estimate, can be big in worst case**

# (Naive) K-Means Disadvantages

- Struggles with some cluster shapes, sizes and outliers
- Struggles with categorical data
- Guaranteed convergence only to local (not global optimum)
- Need to specify *k* in advance
- “Curse of dimensionality”
- High worst-case time complexity

# K-Means Advantages

- **Simple to implement**
- **Good performance in many practical scenarios**
- **Adaptations and combinations can handle outliers, different shapes and sizes, higher dimensions, categorical data…**
  Examples: k-medians, k-modes, k-medoids, hierarchical clusterings, kernel methods, dimensionality reduction…
- **Performance can be improved by non-naive implementations**
  Example: using k-d trees to select initial centroids

# Thank you!

www.rodrigocanaan.com
rocanaan@gmail.com
@rocanaan

Scholar Page