Clustering Analysis I - Basic Concepts

by @chenwang

Table of Content

Part 1: Understanding Clustering Analysis

Clustering Analysis: Definition and Use Cases

> *Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups.*
>
> —- *Wikipedia*

Exploration: Better understanding of the world

- Biology taxonomy: Hierarchical clustering based on genetic information
- Business: Customer segmentation: potential customers
- Document clustering

Find the most representative cluster prototypes

- Compression

Illustrative Use Case: Movie Clustering

- Each movie is described with 2 features: **# of Kisses** & **# of Kicks**
- Objectives: all movies are classified into 2 categories: or

Movie title	# of kicks	# of kisses	Label
California Man	3	104	?
He's Not Really into Dudes	2	100	?
Beautiful Woman	1	81	?
Kevin Longblade	101	10	?
Robo Slayer 3000	99	5	?
Amped II	98	2	?

Movie Data Generation

```
def generat_movieData():
    dataPoints = array([[3,140],[2,100],[1,81],[101,10],[99,5],[98,2]])
    labels = ['Romance','Romance','Romance','Action','Action','Action']
    return dataPoints, labels
```

Part 2: K-Means

How It Works?

Lloyd’s algorithm

- Find **K** unique clusters
  - each cluster is described by a single point known as the **centroid**
- The center of each cluster is the **MEAN** of the values in that cluster

How to stop: In other words, it repeats until the centroids do not move significantly

Python Implementation (1/2)

1. Randomly select K points as initial centroid

```
def random_centroid (dataPoints, k): # TODO: K 个必须是不同的points

    indices = np.random.randint(len(dataPoints), size=k)

    return dataPoints[indices]
```

2. Calculate the Euclidean distance between to vectors

```
def euclidean_distance(vecA, vecB):
    return np.sqrt(np.sum(np.power(vecA - vecB, 2)))
```

Python Implementation (2/2)

```
def kmeans(dataSet, k, distance_measure=euclidean_distance, centroid_init=r
    num_points = len(dataSet)
    clusterAssment = mat(zeros((num_points,2)))
    centroids = centroid_init(dataSet, k)
    clusterChanged = True # stopping flag

    while clusterChanged:
        clusterChanged = False
        history = np.zeros((num_points),dtype=int)
        for i in range(num_points): # 遍历每个点
            minDist = inf; minIndex = -1
            for j in range(k): # 计算每个点和每个cluster 中心的距离
                distJI = distance_measure(centroids[j,:],dataSet[i,:])
                if distJI < minDist:
                    minDist = distJI; minIndex = j # 第i个点分配到第j个cluste
            if clusterAssment[i,0] != minIndex:
                clusterChanged = True
            clusterAssment[i,:] = minIndex,minDist**2 # 第i 个节点分配的clust
            history[i] = minIndex
        for cent in range(k): # 更新centroid
            ptsInClust = dataSet[nonzero(clusterAssment[:,0].A==cent)[0]]
            centroids[cent,:] = mean(ptsInClust, axis=0)

    return centroids, clusterAssment, label_histories, centroid_history
```

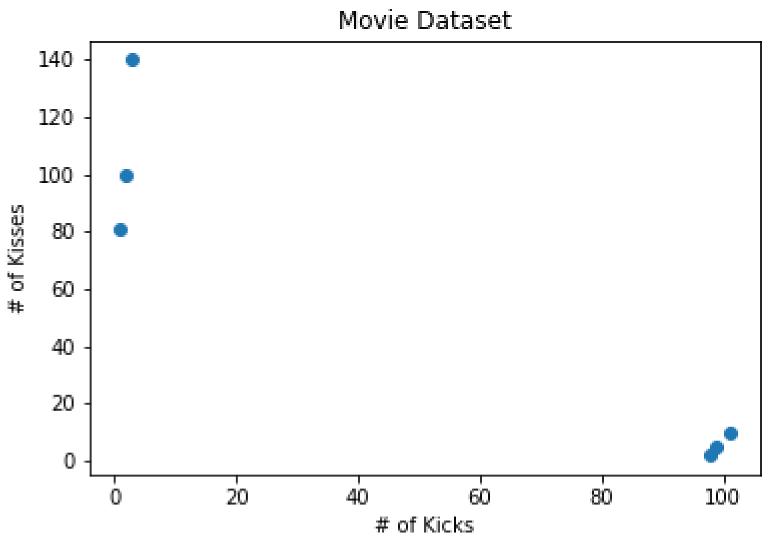
Implementation Using Scikit-Learn

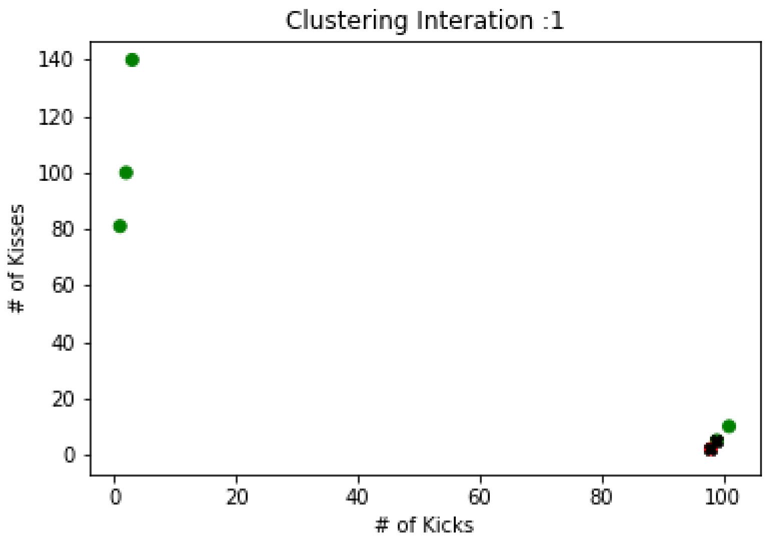
```
from sklearn.cluster import KMeans

kmeans_movie = KMeans(n_clusters=2, random_state=0).fit(movie_data)
```

```
$> kmeans_movie.labels_
array([1, 1, 1, 0, 0, 0], dtype=int32)
```

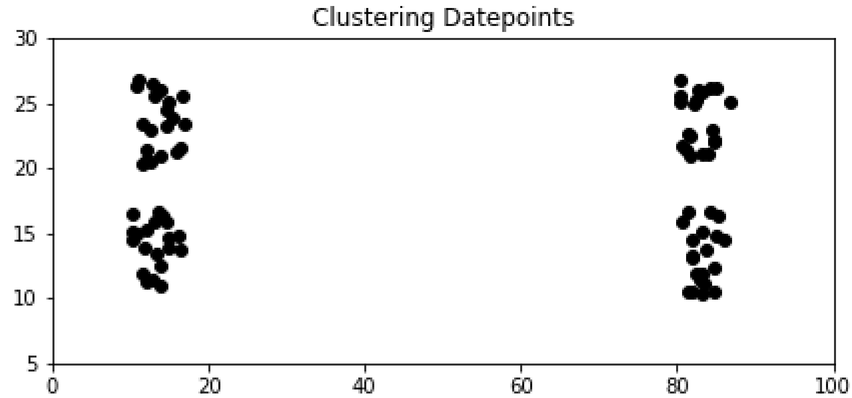
Iterations of Clustering Movie Dataset





Problem: Local Mimimum

Randomly generate some data points in X-Y plane



Running K-Means Clustering (K=4)

Solution 1

Do many times with different initialization of centroids.

Solution 2:

Choose points that are as far as possible.

k-means++ initialization scheme: This initializes the centroids to be (generally) distant from each other, leading to provably better results than random initialization

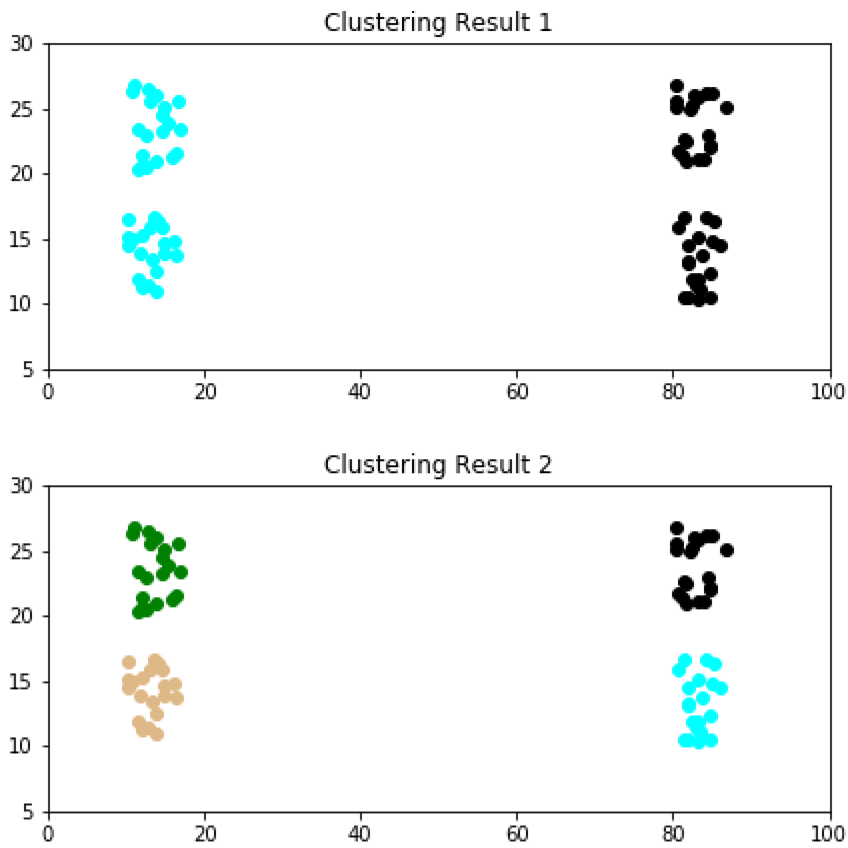
> ```
> init='k-means++'
> ```

"k-means++: The advantages of careful seeding" Arthur, David, and Sergei Vassilvitskii, Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics (2007)

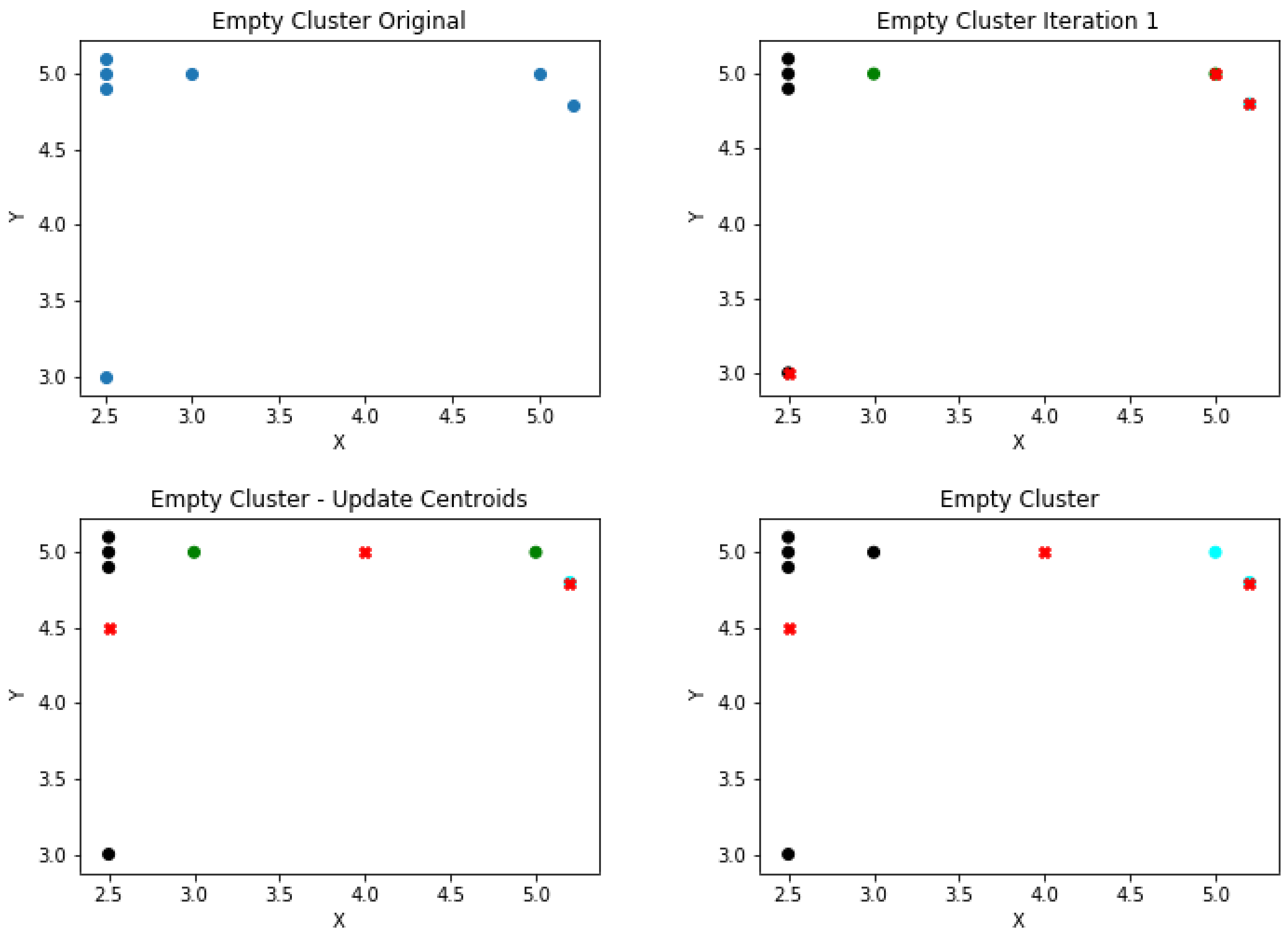
Problem: not scale

How to Choose the Best K?

Q: Which clustering result is more reasonable for you?



Improve Robustness - Handle Empty Cluster



Homework 1

Replace the centroid of an empty cluster.

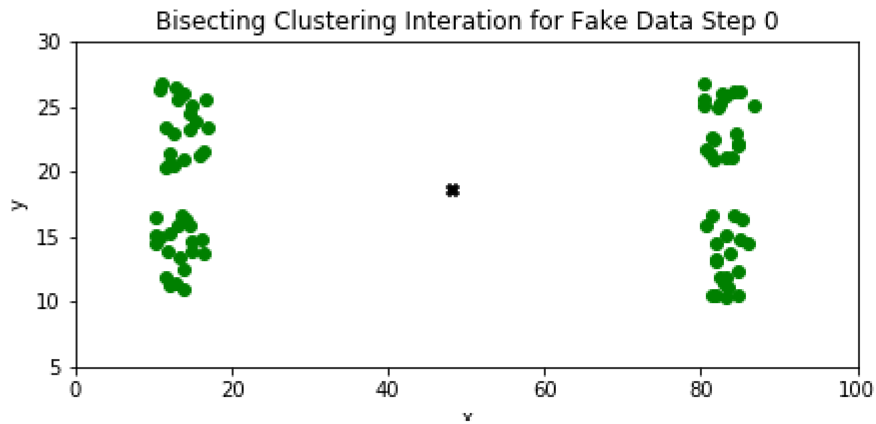
- Solution 1: Select the point that is the farthest from any current centroid;
- Solution 2: Select new centroid for the cluster with the highest SSE;

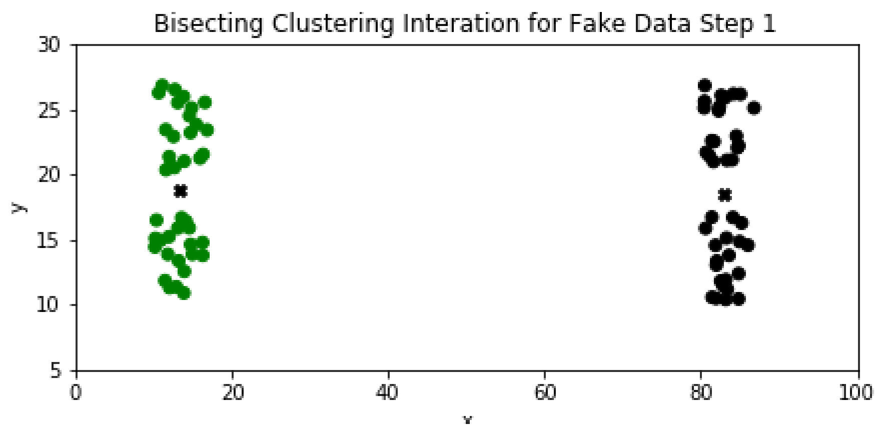
Improve Robustness - Handle Outliers

Part 3: K-Means Variants

Bisecting K-Means

Clustering Using Bisecting K-Means





Part 3: Cluster Evaluation / Validation

Why We Need Clustering Validation?

- Clustering Algorithm can anyhow find out clusters, even if the dataset has no natural cluster structure;

- Determine the **Cluster Tendency**

- Determine the right number of clusters
- Determine how well the clustering results are

backups

clustering: exploratory data analysis