Lecture 9 – K-Means Clustering

Mr. Yousif G. Arshak

University of Zakho

Computer Science Department

yousif.arshak@uoz.edu.krd

OUTLINES

- K-Means Clustering
    - Introduction
    - Algorithm
    - Example
    - Implementation of K means clustering in Python

Introduction

- K-means clustering is a simple unsupervised learning algorithm that is used to solve clustering problems.
- It follows a simple procedure of classifying a given data set into a number of clusters, defined by the letter "k," which is fixed beforehand.
- The clusters are then positioned as points and all observations or data points are associated with the nearest cluster, computed, adjusted and then the process starts over using the new adjustments until a desired result is reached.
- K-means clustering has uses in search engines, market segmentation, data mining, statistics and even astronomy.

Algorithm

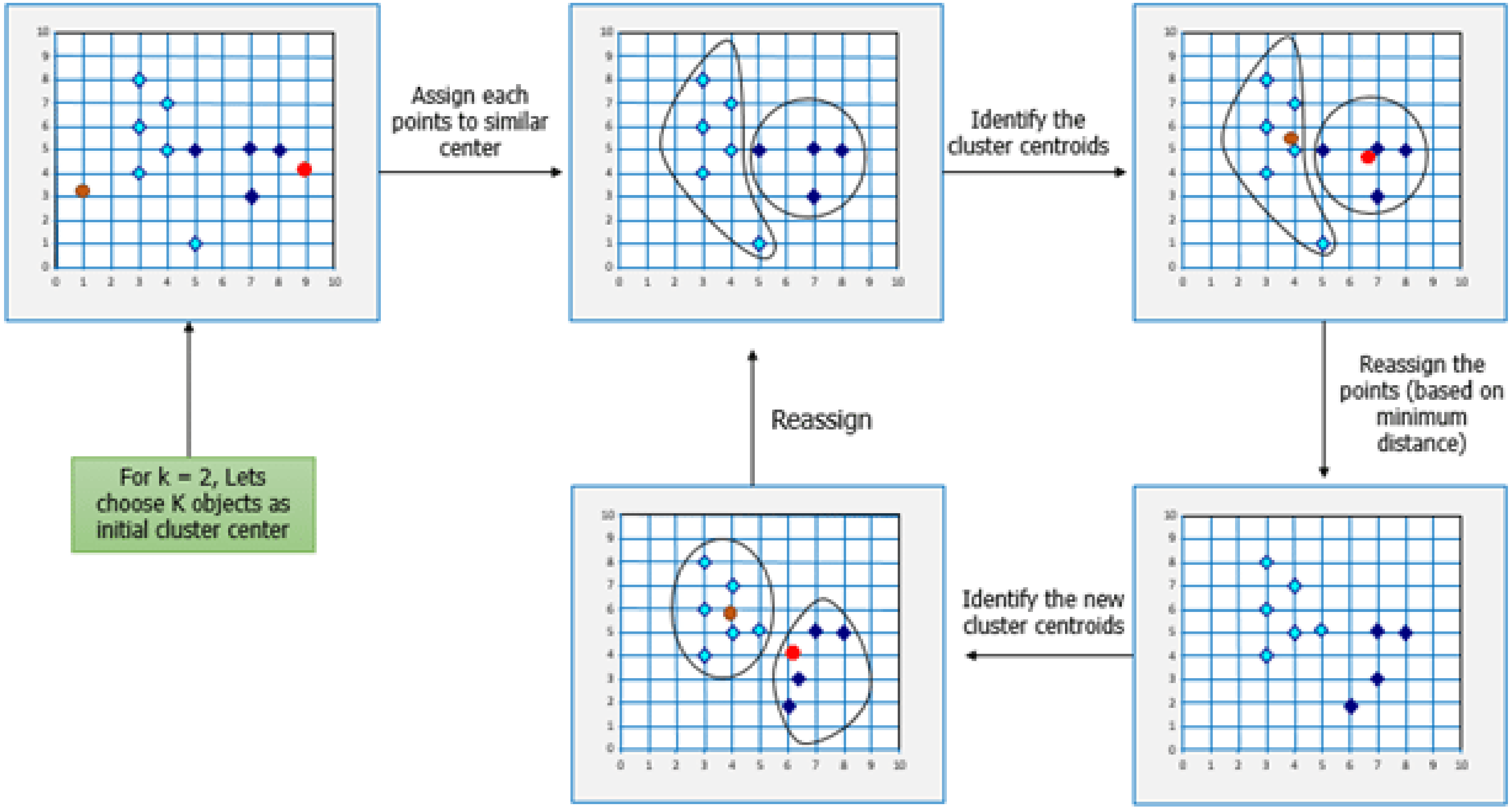
1. K points are placed into the object data space representing the initial group of centroids.
2. Each object or data point is assigned into the closest k.
3. After all objects are assigned, the positions of the k centroids are recalculated.
4. Steps 2 and 3 are repeated until the positions of the centroids no longer move.

Example

- A pizza chain wants to open its delivery centres across a city. What do you think would be the possible challenges?
  - They need to analyze the areas from where the pizza is being ordered frequently.
  - They need to understand as to how many pizza stores has to be opened to cover delivery in the area.
  - They need to figure out the locations for the pizza stores within all these areas in order to keep the distance between the store and delivery points minimum

- K-means Clustering Method:
- If k is given, the K-means algorithm can be executed in the following steps:
  - Partition of objects into k non-empty subsets
  - Identifying the cluster centroids (mean point) of the current partition.
  - Assigning each point to a specific cluster
  - Compute the distances from each point and allot points to the cluster where the distance from the centroid is minimum.
  - After re-allotting the points, find the centroid of the new cluster formed.

The step by step process:



For k = 2, Lets choose K objects as initial cluster center

Assign each points to similar center

Identify the cluster centroids

Reassign the points (based on minimum distance)

Identify the new cluster centroids

Reassign

Implementation of K means clustering in Python

```
#First, start with importing necessary python packages

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# read Iris dataset from csv file from your PC
dataset = pd.read_csv('iris_clustering.csv')
x = dataset.iloc[:, [1, 2, 3, 4]].values
```

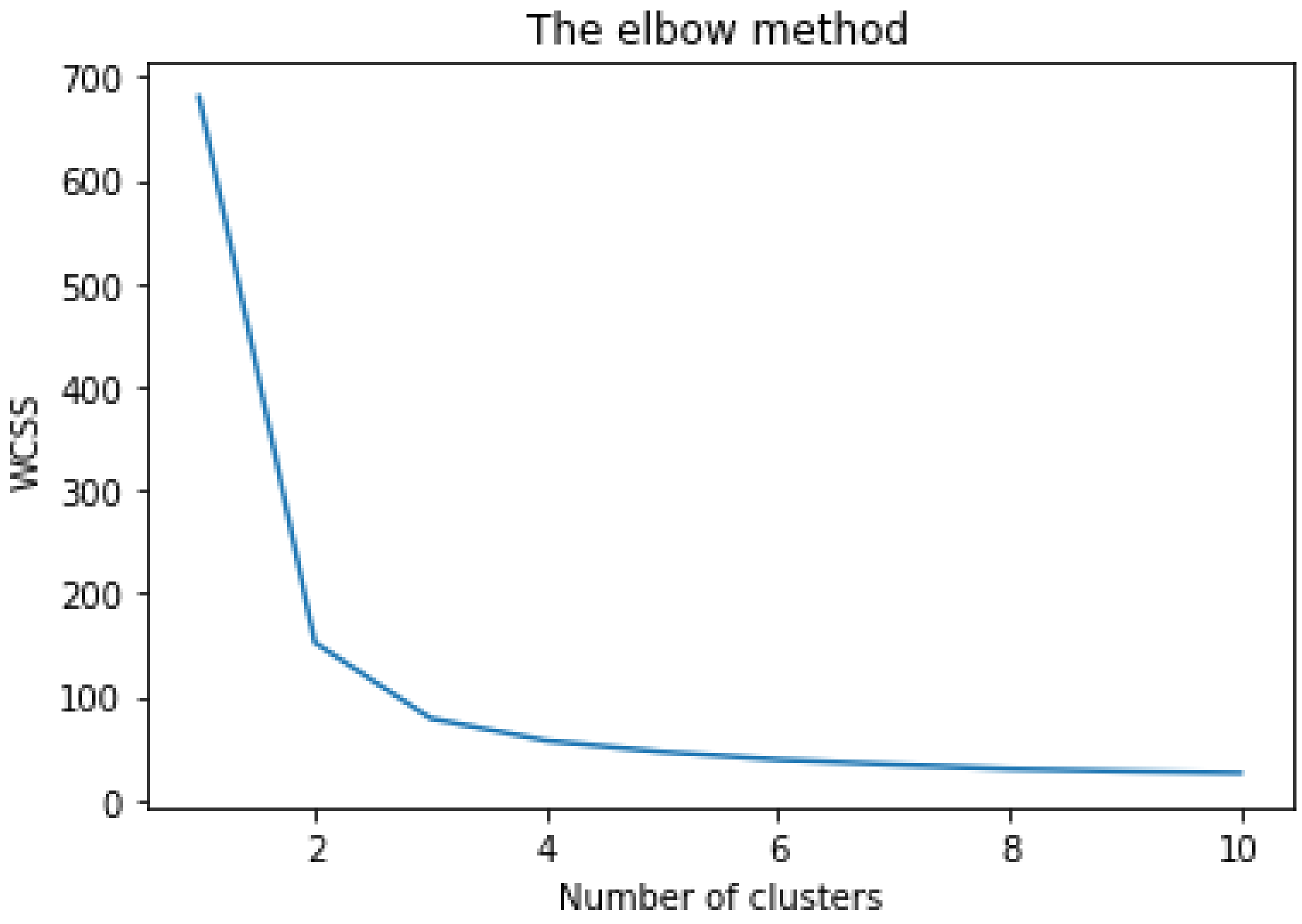
Now we will implement 'The elbow method' on the Iris dataset. The elbow method allows us to pick the optimum amount of clusters for classification. although we already know the answer is 3 it is still interesting to run.

```
#Finding the optimum number of clusters for k-means classification
from sklearn.cluster import KMeans
wcss = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
```

```
kmeans.fit(x)
    wcss.append(kmeans.inertia_)

#Plotting the results onto a line graph, allowing us to observe 'The elbow'
plt.plot(range(1, 11), wcss)
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS') #within cluster sum of squares
plt.show()
```


The elbow method
WCSS
0
100
200
300
400
500
600
700
2
4
6
8
10
Number of clusters

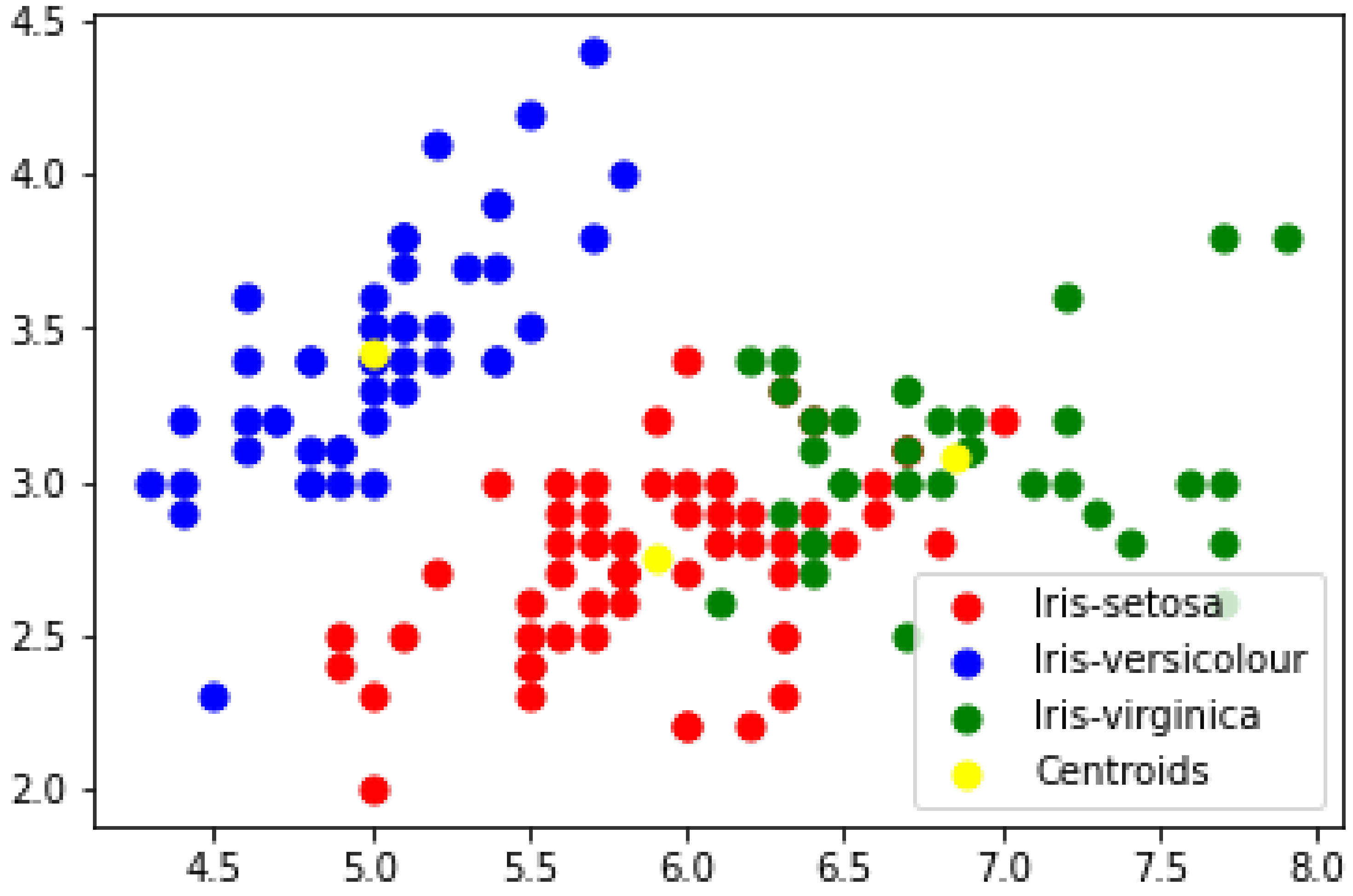
You can clearly see why it is called 'The elbow method' from the above graph, the optimum clusters is where the elbow occurs. This is when the within cluster sum of squares (WCSS) doesn't decrease significantly with every iteration. Now that we have the optimum amount of clusters, we can move on to applying K-means clustering to the Iris dataset.

```
#Applying kmeans to the dataset / Creating the kmeans classifier
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300,
n_init = 10, random_state = 0)
y_kmeans = kmeans.fit_predict(x)
```

```
#Visualising the clusters
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s = 50, c = 'red', label = 'Iris-setosa')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s = 50, c = 'blue', label = 'Iris-versicolour')
plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1], s = 50, c = 'green', label = 'Iris-virginica')

#Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s = 50, c = 'yellow', label = 'Centroids')

plt.legend()
```


4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
● Iris-setosa
● Iris-versicolour
● Iris-virginica
● Centroids