K-means clustering from scratch with Python

Marcelo Fernandes
Wellington, New Zealand
https://marcelofern.com

I. Introduction

K-means clustering is a data-clustering algorithm where the number of clusters is pre-selected by the user. The algorithm will then find ”K” different clusters that split the data based on a definition of distance.

The algorithm can be broken down into the following steps:

1) Select ”K” random data points to be the center of the initial clusters. These centers will be called ”centroids”.
2) For each data point in the dataset, find the closest centroid based on a definition of distance. The Euclidean distance is a common choice.
3) Calculate the new centroids by finding the average point in a given cluster.
4) Repeat steps 2. and 3. until the algorithm converges, i.e., the new centroids are the same as the ones from the previous interaction.

March 28, 2021

II. Example

In this example we will calculate and plot each step of the K-means clustering algorithm for K=3 and the following dataset:

TABLE I
EXAMPLE DATASET

Datapoint	X	Y
A1	2	10
A2	2	5
A3	8	4
A4	5	8
A5	7	5
A6	6	4
A7	1	2
A8	4	9

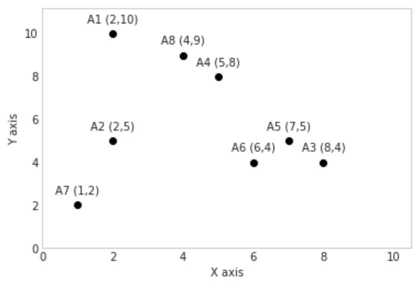

Fig. 1. Dataset in a 2d plot

III. Selecting 3 random data points to be the initial centroids

Three data points can be randomly selected using the following Python code:

```
import random
# pick 3 numbers from 1 to 9 (not inclusive)
indexes = random.sample(range(1, 9),3)
indexes
>>> [1, 4, 7]
```

This means that A1, A4, and A7 have been selected to be the initial centroids.

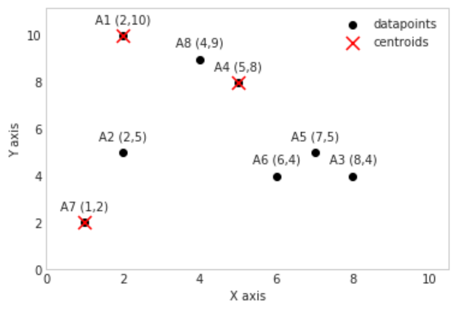

Fig. 2. Initial centroids

IV. Finding which points belong to each cluster

We calculate the Euclidean distance from each data point to each centroid. A point will belong to the closest cluster.

The distance between the points and the centroids will be calculated using the Euclidean distance formula.

```
def distance(v1, v2):
    """
    Returns the Euclidean distance between two
    2d vectors.
    """
    x_distance = (v1[0] - v2[0])**2
    y_distance = (v1[1] - v2[1])**2
    return (x_distance + y_distance)**0.5
```

The initial clusters are showed in Fig.3

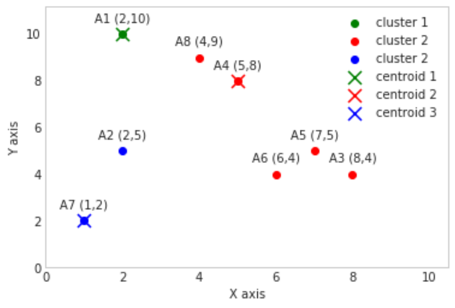

Fig. 3. Initial clusters

V. Calculate the New Centroids

New centroids are calculated by finding the average point for a given cluster:

```
def new_centroid(cluster):
    length = len(cluster)
    mean_x = sum([x for x, _ in cluster])/length
    mean_y = sum([y for _, y in cluster])/length
    return [mean_x, mean_y]
```

The new centroids will therefore be:

```
centroid_1 = [2.0, 10.0]
centroid_2 = [6.0, 6.0]
centroid_3 = [1.5, 3.5]
```

We are now going to repeat the last two steps until the algorithm reaches convergence.

VI. Second Interaction

New centroids:

```
centroid_1 = [3.0, 9.5]
centroid_2 = [6.5, 5.25]
centroid_3 = [1.5, 3.5]
```

VII. Third Interaction

New centroids:

```
centroid_1 = [3.6666666666666665, 9.0]
centroid_2 = [7.0, 4.333333333333333]
centroid_3 = [1.5, 3.5]
```

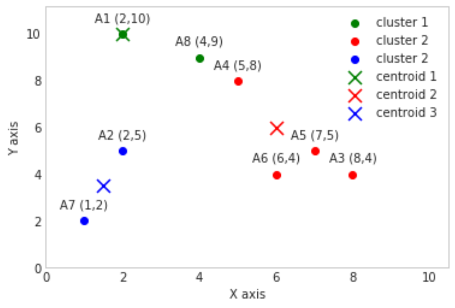

Fig. 4. New centroids, second interaction

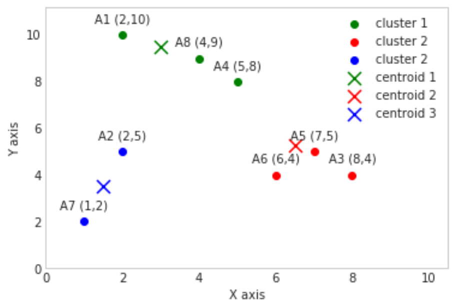

Fig. 5. New centroids, third interaction

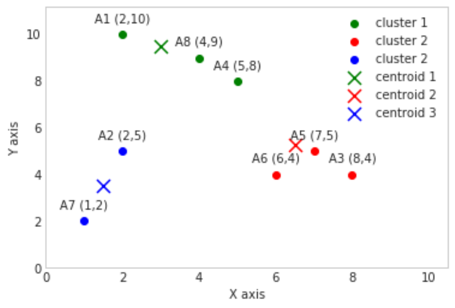

Fig. 6. Final clusters

VIII. Forth Interaction (Convergence)

New centroids:

```
centroid_1 = [3.6666666666666665, 9.0]
centroid_2 = [7.0, 4.333333333333333]
centroid_3 = [1.5, 3.5]
```

IX. Appendix

A. Code for generating Fig. 1:

```
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')

x_coordinates = [2,2,8,5,7,6,1,4]
y_coordinates = [10,5,4,8,5,4,2,9]
points = [
    'A1','A2','A3','A4',
    'A5','A6','A7','A8'
]

# graph configuration
plt.grid(b=None)
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.xlim([0,10.5])
plt.ylim([0,11.2])

# labels
iterator = zip(
    points,
    x_coordinates,
    y_coordinates
)
for point, x, y in iterator:
    label = f"{point} ({x},{y})"
    plt.annotate(
        label,
        (x,y),
        textcoords="offset points",
        xytext=(0,10),
        ha='center'
    )
fig = plt.plot(
    x_coordinates,
    y_coordinates, 'o', color='black'
)
```

B. Code for generating Fig. 2:

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('seaborn-whitegrid')

x_coordinates = [2,2,8,5,7,6,1,4]
y_coordinates = [10,5,4,8,5,4,2,9]
centroids_x = [2,5,1]
centroids_y = [10,8,2]
points = [
    'A1','A2','A3','A4',
    'A5','A6','A7','A8'
]

# plot!
fig = plt.figure()
ax = fig.add_subplot(111)

# graph configuration
plt.grid(b=None)
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.xlim([0,10.5])
plt.ylim([0,11.2])

iterator = zip(
    points,
    x_coordinates,
    y_coordinates
)
for point, x, y in iterator:
    label = f"{point} ({x},{y})"
    ax.annotate(
        label,
        (x,y),
        textcoords="offset points",
        xytext=(0,10),
        ha='center'
    )

ax.scatter(
    x_coordinates, y_coordinates, marker='o',
    color='black', label='datapoints'
)
ax.scatter(
    centroids_x, centroids_y, marker='x',
    color='red', label='centroids', s=120
)
ax.legend()
```

C. Code for generating Fig. 3:

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('seaborn-whitegrid')

points_names = [
    'A1','A2','A3','A4',
    'A5','A6','A7','A8'
]

data_points = [
    [2,10], [2,5], [8,4],
    [5,8], [7,5], [6,4],
    [1,2], [4,9]
]
x_coordinates = [point[0] for point in data_points]
y_coordinates = [point[1] for point in data_points]

centroid_1 = [2,10]
centroid_2 = [5,8]
centroid_3 = [1,2]

centroids = [centroid_1, centroid_2, centroid_3]

cluster_1 = []
cluster_2 = []
cluster_3 = []

def distance(v1, v2):
    return (
        (v1[0] - v2[0])**2
        + (v1[1] - v2[1])**2
    )**0.5

for point in data_points:
    dist_c1 = distance(point, centroids[0])
    dist_c2 = distance(point, centroids[1])
    dist_c3 = distance(point, centroids[2])
    min_distance = min([dist_c1, dist_c2, dist_c3])
    if min_distance == dist_c1:
        cluster_1.append(point)
    elif min_distance == dist_c2:
        cluster_2.append(point)
    else:
        cluster_3.append(point)

cluster_1_x = [point[0] for point in cluster_1]
cluster_2_x = [point[0] for point in cluster_2]
cluster_3_x = [point[0] for point in cluster_3]

cluster_1_y = [point[1] for point in cluster_1]
cluster_2_y = [point[1] for point in cluster_2]
cluster_3_y = [point[1] for point in cluster_3]

# initiate the plot
fig = plt.figure()
ax = fig.add_subplot(111)

# graph configuration
plt.grid(b=None)
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.xlim([0,10.5])
plt.ylim([0,11.2])


# labels
iterator = zip(
    points_names,
    x_coordinates,
    y_coordinates
)
for point_name, x, y in iterator:
    label = f"{point_name} ({x},{y})"
    ax.annotate(
        label,
        (x,y),
        textcoords="offset points",
        xytext=(0,10),
        ha='center'
    )

# plot the data
ax.scatter(
    cluster_1_x, cluster_1_y, marker='o',
    color='green', label='cluster 1'
)
ax.scatter(
    cluster_2_x, cluster_2_y, marker='o',
    color='red', label='cluster 2'
)
ax.scatter(
    cluster_3_x, cluster_3_y, marker='o',
    color='blue', label='cluster 2'
)
ax.scatter(
    centroid_1[0], centroid_1[1], marker='x',
    color='green', label='centroid 1', s=120
)
ax.scatter(
    centroid_2[0], centroid_2[1], marker='x',
    color='red', label='centroid 2', s=120
)
ax.scatter(
    centroid_3[0], centroid_3[1], marker='x',
    color='blue', label='centroid 3', s=120
)
ax.legend()
```

D. *Code for generating Fig. 6:*

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('seaborn-whitegrid')

points_names = [
    'A1','A2','A3','A4',
    'A5','A6','A7','A8'
]

data_points = [
    [2,10], [2,5], [8,4], [5,8],
    [7,5], [6,4], [1,2], [4,9]
]
x_coordinates = [point[0] for point in data_points]
y_coordinates = [point[1] for point in data_points]

centroid_1 = [3.6666666666666665, 9.0]
centroid_2 = [7.0, 4.333333333333333]
centroid_3 = [1.5, 3.5]

centroids = [centroid_1, centroid_2, centroid_3]

cluster_1 = []
cluster_2 = []
cluster_3 = []

def distance(v1, v2):
    return ( (v1[0] - v2[0])**2 + (v1[1] - v2[1])**2
     )**0.5

for point in data_points:
    dist_c1 = distance(point, centroids[0])
    dist_c2 = distance(point, centroids[1])
    dist_c3 = distance(point, centroids[2])
    min_distance = min([dist_c1, dist_c2, dist_c3])
    if min_distance == dist_c1:
        cluster_1.append(point)
    elif min_distance == dist_c2:
        cluster_2.append(point)
    else:
        cluster_3.append(point)

cluster_1_x = [point[0] for point in cluster_1]
cluster_2_x = [point[0] for point in cluster_2]
cluster_3_x = [point[0] for point in cluster_3]

cluster_1_y = [point[1] for point in cluster_1]
cluster_2_y = [point[1] for point in cluster_2]
cluster_3_y = [point[1] for point in cluster_3]

# initiate the plot
fig = plt.figure()
ax = fig.add_subplot(111)

# graph configuration
plt.grid(b=None)
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.xlim([0,10.5])
plt.ylim([0,11.2])


# labels
iterator = zip(
    points_names,
    x_coordinates,
    y_coordinates
)
for point_name, x, y in iterator:
    label = f"{point_name} ({x},{y})"
    ax.annotate(
        label,
        (x,y),
        textcoords="offset points",
        xytext=(0,10),
        ha='center'
    )

# plot the data
ax.scatter(
    cluster_1_x, cluster_1_y, marker='o',
    color='green', label='cluster 1'
)
ax.scatter(
    cluster_2_x, cluster_2_y, marker='o',
    color='red', label='cluster 2'
)
ax.scatter(
    cluster_3_x, cluster_3_y, marker='o',
    color='blue', label='cluster 2'
)
ax.scatter(
    centroid_1[0], centroid_1[1], marker='x',
    color='green', label='centroid 1', s=120
)
ax.scatter(
    centroid_2[0], centroid_2[1], marker='x',
    color='red', label='centroid 2', s=120
)
ax.scatter(
    centroid_3[0], centroid_3[1], marker='x',
    color='blue', label='centroid 3', s=120
)
ax.legend()

def new_centroid(cluster):
    length = len(cluster)
    mean_x = sum([x for x, _ in cluster])/length
    mean_y = sum([y for _, y in cluster])/length
    return [mean_x, mean_y]

print(new_centroid(cluster_1))
print(new_centroid(cluster_2))
print(new_centroid(cluster_3))
```