Artificial Intelligence: Clustering

Marián Kurčina

ZS 2024

xkurcinam@stuba.sk
marian.kurcina2003@gmail.com

Table of Contents

Task

We have a canvas, 10,000x10,000 with coordinates from -5000 to +5000. The space will initially be filled with 20 random points. After they are generated, another 20,000 points will be generated by selecting a random point from all previously generated points and moving it by an offset on the x and y axes. The offset will be a random number from -100 to 100, and if a point is closer than 100 to the edge, the offset will be adjusted so that the point is not generated outside the canvas.
The task is to program a clustering algorithm that divides the space into k clusters using:

- Agglomerative clustering, where the center is the centroid
- Agglomerative clustering, where the center is the medoid

The condition for the clusters is that none of them can have an average distance of their points from the center greater than 500.
It is necessary to plot the resulting area and distinguish the individual clusters.

Agglomerative Clustering

Agglomerative Clustering is a clustering method where each point starts as its own cluster, and clusters are progressively merged based on their mutual distance. After each merge, the new distances between the clusters are recalculated, and a new center for the cluster is determined. The center of a cluster can be either a centroid or a medoid. Clusters continue to merge until a pre-set condition is violated.

Centroid

It is the arithmetic mean of all the points in the cluster. This point does not have to be a point of the cluster.

Medoid

It is a point of the cluster that has the smallest average distance to all other points in the cluster. It is also the closest point to the centroid. Unlike the centroid, the medoid is a point within the cluster.

Solution description

At the beginning, 20 random points will be generated in the space. Subsequently, 20,000 points will be created from the already existing points. When creating these points, a point will be chosen first, from which a new point will be generated. If this point is closer than 100 to the edge of the canvas, the offset will be adjusted so that the point is not created outside the canvas, with the maximum offset being the distance from the edge of the canvas. If the point is more than 100 away from the edge, the offset will be a number between -100 and 100. Separate offsets will be created for x and y. After the points are generated, distances between all points will be calculated and stored in a matrix.
The actual clustering will proceed through repeated selection of the shortest distance from the

distance matrix, checking whether the condition will be violated after merging these clusters, merging the clusters, and updating the distance matrix. This cycle will repeat until the condition is violated.
The condition check will work by finding the center of the cluster and calculating the average distance of all its points from the center. If this average is greater than 500, the cycle will be interrupted, and the clusters will not be merged.
If the condition is not violated, the clusters will merge. All points from one cluster will be moved to the other, and the information about the clusters will be updated, including the new center and distances to the newly formed cluster.
After the condition is violated, the resulting clusters will be visualized on the canvas. Each cluster will have a random color to distinguish them.
I will program the solution in Python using the PyPy compiler. I will use the random, plotly, and math libraries.

Installation and Usage Guide for PyPy

On the page https://pypy.org/download.html, download the version of PyPy compatible with your operating system. After the installation is complete, create a new environment variable named Path and assign it the value containing the path to the installed pypy.exe file. Then, in the project settings, set the project interpreter to the pypy.exe file.

Code description

At the beginning, the data necessary for clustering will be initialized, then the points will be created, the initial distance matrix will be calculated, and subsequently, the cluster merging cycle will take place. After the cycle ends, the result will be plotted.

Data Representation

The data I will be using for clustering are:

- Number of initial points (user input) - first_num
- Total number of points (user input) - total_num
- Clustering type (user input) - medoid
    - Centroid/Medoid
- Coordinates of the points – points_x, points_y
- Current number of clusters - cluster_num
- Content of the clusters - clusters
    - List of points in each cluster
- Coordinates of cluster centers - clusters_x, clusters_y
- Distance matrix of cluster centers - distances

Functions description

create_points

The function first generates initial points with random coordinates between -5000 and 5000, and these coordinates are stored in the points_x and points_y arrays. Then, the remaining points are generated by randomly selecting the index of a previous point and creating a new point based on its coordinates using x_offset and y_offset.

If the coordinate is between -4900 and 4900 (i.e., the point is more than 100 units away from the edge), the offset is between -100 and 100. If the coordinate is between -5000 and -4900, the offset is a value between the distance to the edge (a negative value) and 100. If the coordinate is between 4900 and 5000, the offset is between -100 and the distance to the edge. This process repeats until all points are generated.

Afterward, the information about the points is copied into the clusters_x and clusters_y arrays. The clusters array is populated with points at their corresponding indices, with each point initially being its own cluster.

calculate_initial_distances

The function calculates the distance from each point to every other point using the Pythagorean theorem.

find_min

The function traverses the distance matrix and returns the indices of the clusters with the smallest distance, where the first index is smaller than the second.

average

The function calculates the average distance of cluster points from the cluster center and compares it to 500. If the average is less than 500, it returns True; if it is greater, it returns False.
At the beginning, the points from both clusters are merged, and their total number is calculated. Then, the coordinates are summed, and the average is calculated, which gives the centroid. If using a medoid, the closest point to the centroid is found by calculating the distance of each point from the centroid and searching for the smallest distance. The index of the point with the smallest distance is saved. After processing all the points, the closest point to the centroid becomes the cluster center.
Once the center is found, the distances of all points from it are calculated, and the average distance is computed.
If the average distance is less than 500, the condition is not violated. If it is greater than 500, the condition is violated, and the clusters will not be merged.

add_cluster

The function is used to merge cluster A and cluster B. The points from cluster B will be added to cluster A, and the contents of cluster B will be cleared afterward.

calculate_centroid

The function is used to calculate the centroid. It computes the sum of the coordinates of the points and returns their average.

find_medoid

The function is used to find the point closest to the centroid, i.e., it finds the medoid. The function iterates through the cluster's points, calculating the distance from each point to the centroid and looking for the smallest distance. After checking all the points, the function returns the coordinates of the medoid.

update_distances

The function is used to update the distance matrix after merging two clusters. It calculates the distances from the newly formed cluster to the remaining clusters. Then, it swaps the data of the last valid cluster (at index clusters_num - 1) and the second cluster that was merged. Finally, the number of clusters is decreased by 1, making the data of the second cluster invalid.

merge

The function is responsible for calling the other functions and thus directly manages the merging of clusters. It first finds the shortest distance between clusters using find_min. Then, it checks whether merging these clusters meets the condition. If the condition is not met, the function returns False. If the condition is satisfied, it calls add_cluster and calculates the center of the new cluster using calculate_centroid. If using the medoid, it finds its coordinates using find_medoid. The coordinates of the cluster center are stored in clusters_x and clusters_y. Finally, the distances in the distance matrix are updated, and the function returns True.

draw

The function is responsible for visualizing the clustering results. First, traces are created for the clusters. For each cluster, a color is generated based on the cluster's order, and the clusters are visualized using the clusters array. Then, a trace is created for displaying the centroids or medoids. At the end of the function, the layout of the canvas is defined, and the canvas is displayed.

Evaluation of the results

First, I will present the test results of the program for different numbers of points, considering both centroid and medoid calculations. Calculating with the centroid was generally slower, which could be due to the program needing to work with decimal numbers, which is more time and memory-intensive. This time difference becomes more apparent as the number of points increases.

	Centroid	Medoid
2 020 points	3.07 seconds	3.02 seconds
5 020 points	33.56 seconds	34.13 seconds
10 020 points	4 minutes 12.49 seconds	4 minutes 7.43 seconds
15 020 points	13 minutes 40.81seconds	13 minutes 38.73 seconds
20 020 points	31 minutes 25.1 seconds	30 minutes 54.46 seconds
25 020 points	61 minutes 11.44 seconds	60 minutes 41.51 seconds

Table 1: Table displaying clustering times with different numbers of points, using either the centroid or medoid.

In the solution with the centroid, it is possible that the result will be two different clusterings. However, as the number of points increases, the likelihood of the medoid creating different clusters decreases. It is more likely that the centroid will be close to the cluster point, and therefore the medoid will not make a difference in the clustering. This difference is noticeable with smaller numbers of points.

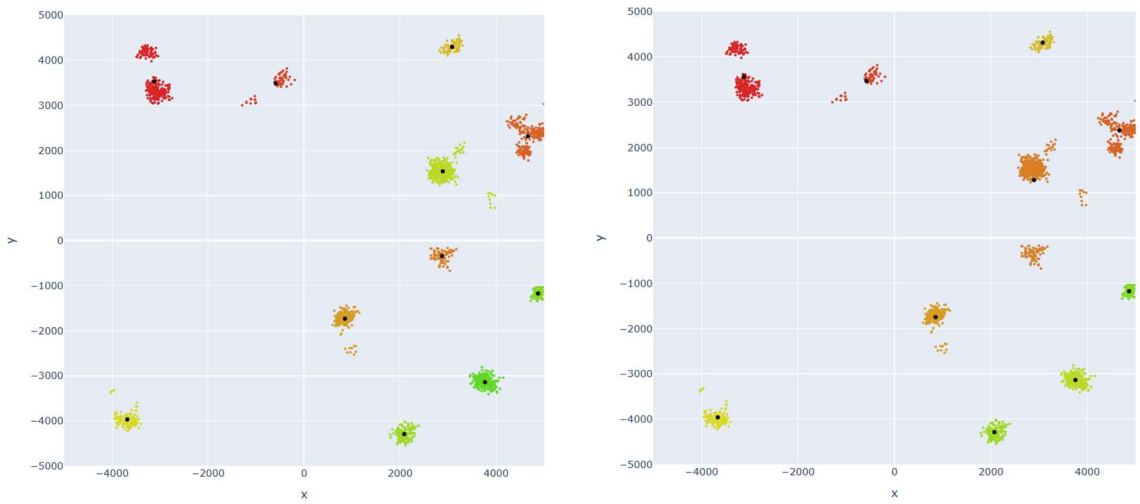

Image 1: Comparison of clustering using centroid and medoid for 2020 points

Agglomerative clustering is effective for small numbers of points, but as the number of initial points increases, the solution time increases exponentially. I can confirm this fact with my measurements, as the initial mergers took several times longer than the subsequent ones. For 20,020 points, the use of centroids or medoids did not change the final solution, but it did change the clustering indices (Image 2 and Image 3).

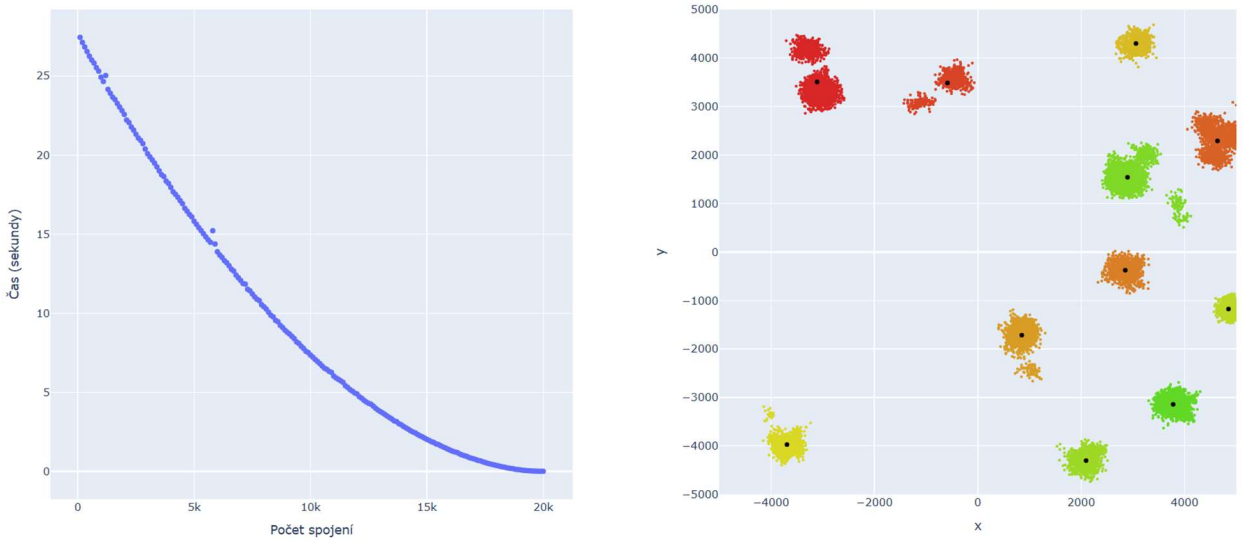

Image 2: Graph showing the development of the time required to merge 100 clusters and the resulting cluster division using the centroid method

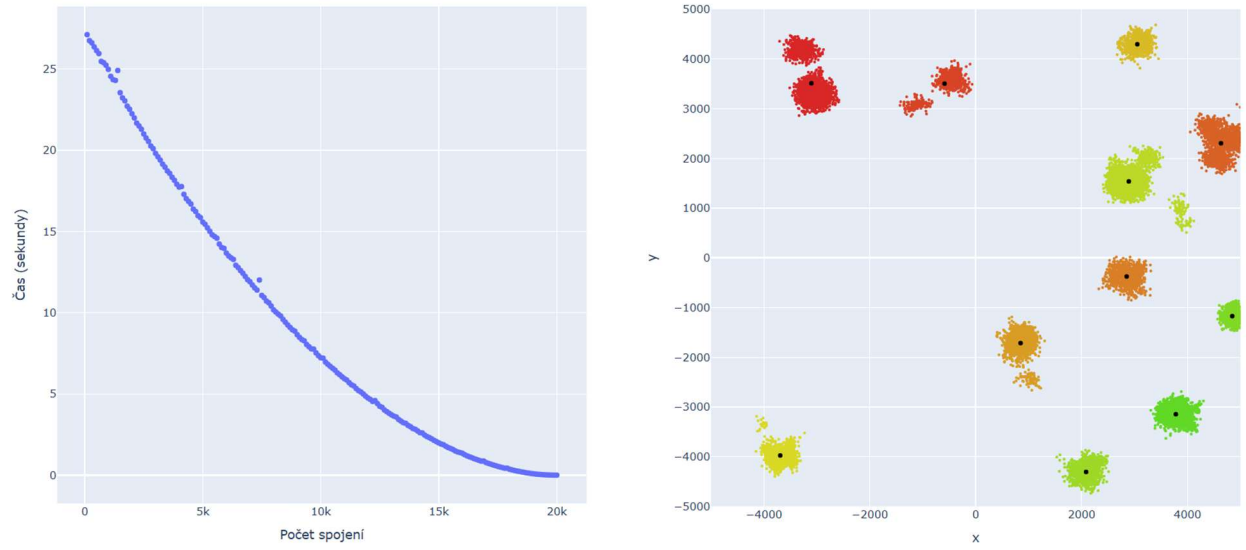

Image 3: Graph showing the development of the time required to merge 100 clusters and the resulting cluster division using the medoid method.

The program generates a different arrangement of points each time, so the examples of visualization for 20,020 points using centroids and medoids are shown below (Image 4).

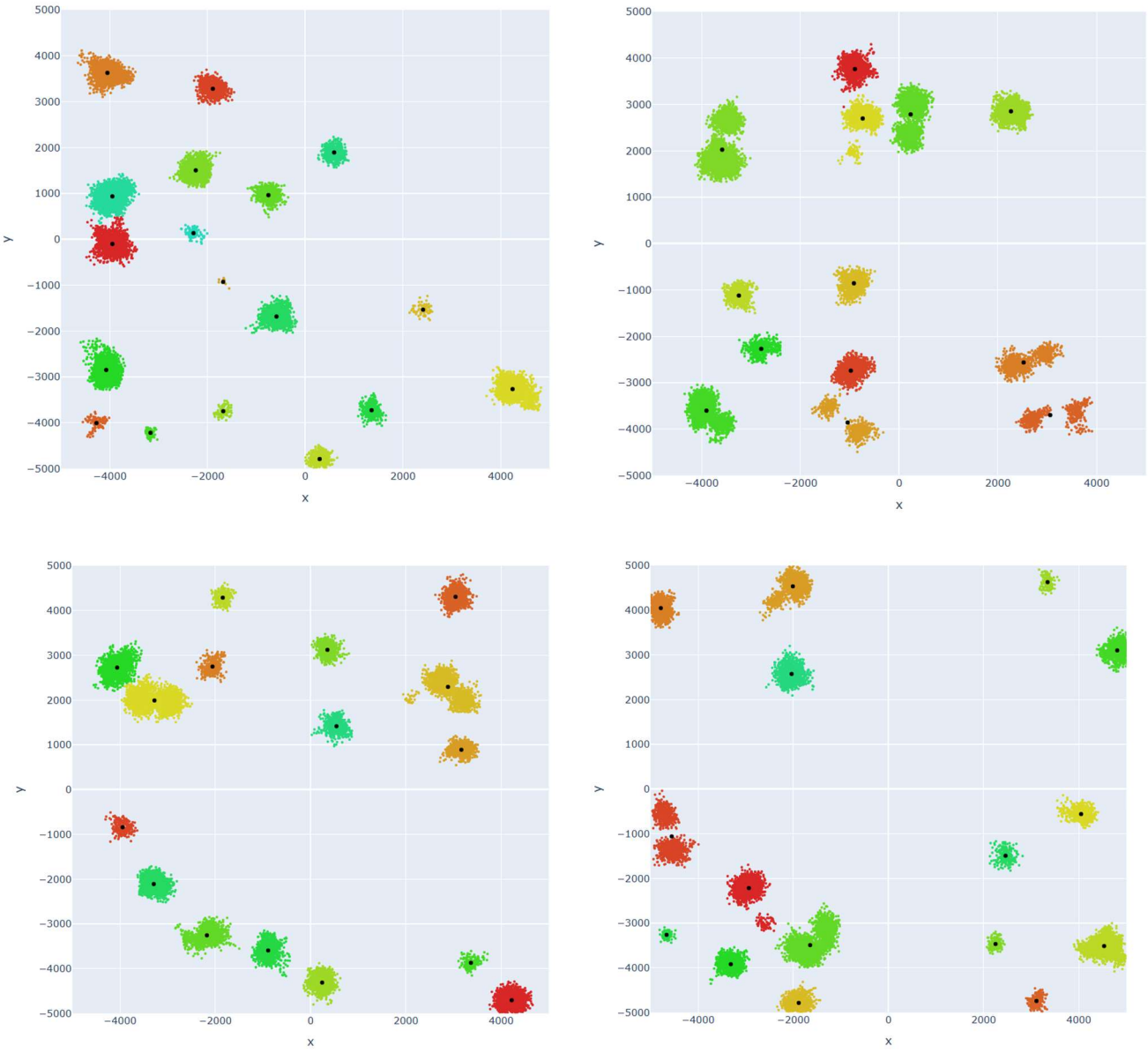

Image 4: Visualization of several attempts with 20,020 points, with solutions using centroids on the left and solutions using medoids on the right.

Conclusion

I managed to implement a program for clustering points into clusters using agglomerative clustering with either a centroid or a medoid. I identified key parts that make the program run faster or slower, and I was able to identify and compare the program's behavior for different numbers of initial points and how it behaves when using medoids and centroids. I summarized and visualized the results of the observations in the documentation.