

K-means clustering

«Project 1: Testing and Documentation»

Maryia Pilipchuk

2026

Task

Implementation of the K-Means clustering algorithm to group N points with D dimensions into k clusters. The program must read data from an input file, process it, and output the group indices for all points.

Analysis

The K-Means algorithm is an iterative process used to partition data into k distinct, non-overlapping clusters.

Initialization: The program selects k initial centroids.

Assignment: Each data point is assigned to the nearest centroid based on Euclidean distance.

Update: Centroids are recalculated by taking the mean of all points assigned to that cluster.

Convergence: The process repeats until the centroids no longer change significantly.

Table of Contents

Class Index

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Centroid (Represents a centroid in K-Means clustering)

KMeans (Implements the K-Means clustering algorithm)

Point (Represents a data point in multidimensional space)

File Index

File List

Here is a list of all documented files with brief descriptions:

kmeans.cpp (Implementation of KMeans class)

kmeans.h (Declaration of KMeans class)

main.cpp (Entry point of the K-Means clustering program)

point.h (Defines the Point structure)

Class Documentation

Centroid Struct Reference

Represents a centroid in K-Means clustering.

```
#include <kmeans.h>
```

Public Attributes

- `vector< double > coords`
Coordinates of centroid.

Detailed Description

Represents a centroid in K-Means clustering.

The documentation for this struct was generated from the following file:

kmeans.h

KMeans Class Reference

Implements the K-Means clustering algorithm.

```
#include <kmeans.h>
```

Public Member Functions

- **KMeans** (int k)
Constructor.
- void **loadData** (const string &filename)
Loads dataset from CSV file.
- void **run** ()
Runs the clustering algorithm.
- void **saveResults** (const string &filename)
Saves clustering results to file.

Detailed Description

Implements the K-Means clustering algorithm.

This class loads data, performs clustering, and saves the results to a file.

Constructor & Destructor Documentation

KMeans::KMeans (int k)

Constructor.

Constructor implementation.

Parameters

k	Number of clusters.
-----	---------------------

Member Function Documentation

void KMeans::loadData (const string & filename)

Loads dataset from CSV file.

Parameters

<i>filename</i>	Path to input file.
-----------------	---------------------

void KMeans::run ()

Runs the clustering algorithm.

Executes the K-Means algorithm.

void KMeans::saveResults (const string & filename)

Saves clustering results to file.

Saves clustering results to output file.

Parameters

<i>filename</i>	Path to output file.
-----------------	----------------------

The documentation for this class was generated from the following files:

kmeans.hkmeans.cpp

Point Struct Reference

Represents a data point in multidimensional space.

```
#include <point.h>
```

Public Attributes

- `std::vector< double > coords`
Coordinates of the point.
- `int cluster = -1`
Cluster assignment.

Detailed Description

Represents a data point in multidimensional space.

The documentation for this struct was generated from the following file:

point.h

File Documentation

kmeans.cpp File Reference

Implementation of **KMeans** class.

```
#include "kmeans.h"  
#include <fstream>  
#include <sstream>  
#include <iostream>  
#include <cstdlib>  
#include <cmath>
```

Detailed Description

Implementation of **KMeans** class.

kmeans.h File Reference

Declaration of **KMeans** class.

```
#include <vector>
#include <string>
#include "point.h"
```

Classes

struct **Centroid***Represents a centroid in K-Means clustering.*

class **KMeans***Implements the K-Means clustering algorithm.*

Detailed Description

Declaration of **KMeans** class.

kmeans.h

Go to the documentation of this file.

```
1 #ifndef KMEANS_H
2 #define KMEANS_H
3
4
5
6
7
8
9 #include <vector>
10 #include <string>
11 #include "point.h"
12
13 using namespace std;
14
15
16
17
18 struct Centroid {
19     vector<double> coords;
20 };
21
22
23
24
25
26
27
28
29
30
31 class KMeans {
32 public:
33
34     KMeans(int k);
35
36
37     void loadData(const string& filename);
38
39     void run();
40
41     void saveResults(const string& filename);
42
43 private:
44     int k;
45     int dimensions = 0;
46
47     vector<Point> points;
48     vector<Centroid> centroids;
49
50     void initCentroids();
51
52     void assignClusters();
53
54     void updateCentroids();
55
56     double distance(const Point& p, const Centroid& c);
57
58     bool hasConverged(const vector<Centroid>& oldCentroids, double tol);
59 };
60
61 #endif
62
63
```

main.cpp File Reference

Entry point of the K-Means clustering program.

```
#include <iostream>
#include <string>
#include "kmeans.h"
```

Functions

- void **printManual** ()
Prints program usage instructions.
- int **main** (int argc, char *argv[])
Main function of the program.

Detailed Description

Entry point of the K-Means clustering program.

This program performs K-Means clustering on a dataset provided in CSV format.

Author

Maryia Pilipchuk

Date

2026

Function Documentation

int main (int argc, char * argv[])

Main function of the program.

Parses command-line arguments and runs the K-Means algorithm.

Parameters

<i>argc</i>	Number of command-line arguments
<i>argv</i>	Array of command-line arguments

Returns

int Exit status code

point.h File Reference

Defines the **Point** structure.
`#include <vector>`

Classes

struct **Point***Represents a data point in multidimensional space.*

Detailed Description

Defines the **Point** structure.

point.h

Go to the documentation of this file.

```
1 #ifndef POINT_H
2 #define POINT_H
3
4
5
6
7
8
9 #include <vector>
10
11
12
13
14
15 struct Point {
16
17     std::vector<double> coords;
18
19
20     int cluster = -1;
21 };
22
23
24 #endif
```

External spec. user's manual

The program is a command-line utility. To run the program, use the following syntax:

`Project_1.exe -i <input_file> -o <output_file> -k <number_of_clusters>.`

Command-line switches:

-i: Path to the input CSV file containing N points.

-o: Path to the output TXT file where results will be saved.

-k: The number of clusters (groups) to create.

Example:

`Project_1.exe -i Iris.csv -o out.txt -k 3`

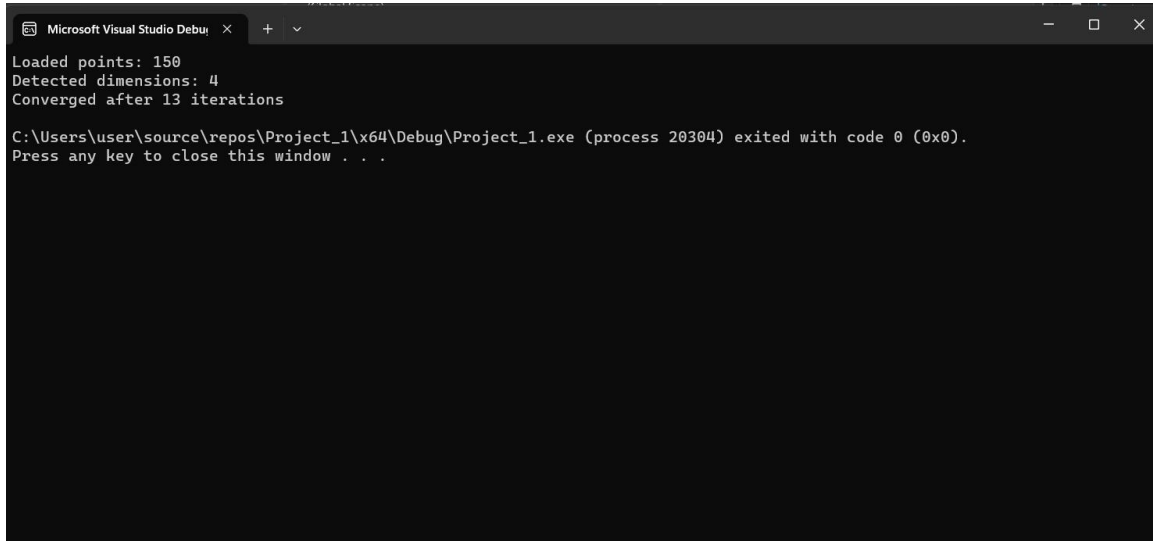
Testing

Test Case 1: Using the Iris.csv dataset.

Input: 150 points with 4 dimensions.

Execution: Running with -k 3.

Result: The program successfully assigns each point to one of the 3 groups and saves the indices in the output file.

A screenshot of a Microsoft Visual Studio Debug console window. The window has a dark background and a title bar that reads "Microsoft Visual Studio Debug". The console output shows the following text: "Loaded points: 150", "Detected dimensions: 4", "Converged after 13 iterations", "C:\\Users\\user\\source\\repos\\Project_1\\x64\\Debug\\Project_1.exe (process 20304) exited with code 0 (0x0).", and "Press any key to close this window . . .".

```
Microsoft Visual Studio Debug
Loaded points: 150
Detected dimensions: 4
Converged after 13 iterations
C:\\Users\\user\\source\\repos\\Project_1\\x64\\Debug\\Project_1.exe (process 20304) exited with code 0 (0x0).
Press any key to close this window . . .
```

File Edit View

K-Means Clustering

Centroids:

Centroid 0: 125 6.57059 2.97059 5.52353

Centroid 1: 25 5.00612 3.42041 1.46531

Centroid 2: 74.5 5.922 2.78 4.206

Points and clusters:

Point 0: (1, 5.1, 3.5, 1.4) -> cluster 1

Point 1: (2, 4.9, 3, 1.4) -> cluster 1

Point 2: (3, 4.7, 3.2, 1.3) -> cluster 1

Point 3: (4, 4.6, 3.1, 1.5) -> cluster 1

Point 4: (5, 5, 3.6, 1.4) -> cluster 1

Point 5: (6, 5.4, 3.9, 1.7) -> cluster 1

Point 6: (7, 4.6, 3.4, 1.4) -> cluster 1

Point 7: (8, 5, 3.4, 1.5) -> cluster 1

Point 8: (9, 4.4, 2.9, 1.4) -> cluster 1

Point 9: (10, 4.9, 3.1, 1.5) -> cluster 1

Point 10: (11, 5.4, 3.7, 1.5) -> cluster 1

Point 11: (12, 4.8, 3.4, 1.6) -> cluster 1

Point 12: (13, 4.8, 3, 1.4) -> cluster 1

Point 13: (14, 4.3, 3, 1.1) -> cluster 1

Point 14: (15, 5.8, 4, 1.2) -> cluster 1

Point 15: (16, 5.7, 4.4, 1.5) -> cluster 1

Point 16: (17, 5.4, 3.9, 1.3) -> cluster 1

Point 17: (18, 5.1, 3.5, 1.4) -> cluster 1

Point 18: (19, 5.7, 3.8, 1.7) -> cluster 1

Point 19: (20, 5.1, 3.8, 1.5) -> cluster 1

Point 20: (21, 5.4, 3.4, 1.7) -> cluster 1

Point 21: (22, 5.1, 3.7, 1.5) -> cluster 1

Point 22: (23, 4.6, 3.6, 1) -> cluster 1

Point 23: (24, 5.1, 3.3, 1.7) -> cluster 1

Point 24: (25, 4.8, 3.4, 1.9) -> cluster 1

Point 25: (26, 5, 3, 1.6) -> cluster 1

Point 26: (27, 5, 3.4, 1.6) -> cluster 1

Point 27: (28, 5.2, 3.5, 1.5) -> cluster 1

Point 28: (29, 5.2, 3.4, 1.4) -> cluster 1

Point 29: (30, 4.7, 3.2, 1.6) -> cluster 1

Point 30: (31, 4.8, 3.1, 1.6) -> cluster 1

Point 31: (32, 5.4, 3.4, 1.5) -> cluster 1

Point 32: (33, 5.2, 4.1, 1.5) -> cluster 1

Point 33: (34, 5.5, 4.2, 1.4) -> cluster 1

Point 34: (35, 4.9, 3.1, 1.5) -> cluster 1

Point 35: (36, 5, 3.2, 1.2) -> cluster 1

Point 36: (37, 5.5, 3.5, 1.3) -> cluster 1

Point 37: (38, 4.9, 3.1, 1.5) -> cluster 1

Point 38: (39, 4.4, 3, 1.3) -> cluster 1

Point 39: (40, 5.1, 3.4, 1.5) -> cluster 1

Point 40: (41, 5, 3.5, 1.3) -> cluster 1

Index

INDEX