Introduction to modern C++

Julien Wintz

October 21, 2020

Contents

1 Introduction

1.1 On Clustering

Simply speaking `K-means` clustering is an algorithm to classify or to group the objects based on attributes/features into K groups. K is a positive integer number. The grouping is done by minimizing the distances between data and the corresponding cluster centroid. The distance that will be used here is the

Euclidean distance.

The algorithm will then find the K groups of data that minimize the following objective function:

$$F = \sum_{i=1}^{K} \sum_{x_j \in S_i} \left((x_j c_i)^t (x_j c_i)\right)$$

where there are K clusters $S_i, i = 1, 2, ..., K$, and c_i is the centroid or mean point of all the points $x_j \in S_i$.

1.2 On Image Segmentation

Since the aim of clustering analysis is to group data in such a way that similar objects are in one cluster and objects of different clusters are dissimilar, the `k-means` clustering algorithm is commonly used in computer vision as a form of image segmentation.

Image segmentation is the process of partitioning a digital image into multiple segments (sets of pixels, also known as image objects). Usually, boundary detection is used in order to further characterise and label the image objects, but simple coloring is often an initial step for advanced algorithms used, e.g., in computer vision or medical imaging.

Segmentation therefore allows one to identify parts of an image.

Using `K-Means` for image segmentation, is one of the most simple `Machine Learning` applications.

2 K-Means clustering put simply

K-means is an unsupervised model so the data is unlabelled. But the model mathematically allocates each data point to a cluster.

Upon initializing the model, we must pre-decide on a number of clusters. Having to do this in advance is a drawback of the model. I'll choose k=2 (aka. 2 clusters) for this example.

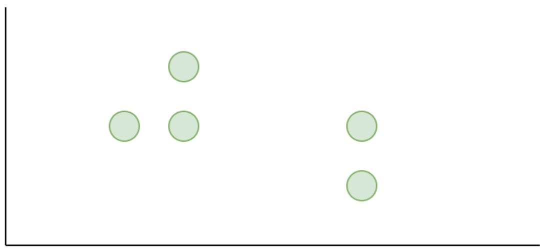

Figure 1: Initialisation: data

2.1 Initialisation

1. Randomly generate an input data set as in Fig. 1.

2.2 Steps

1. Randomly create centroids (cluster centres) in the same vector space as the data (Fig. 2).

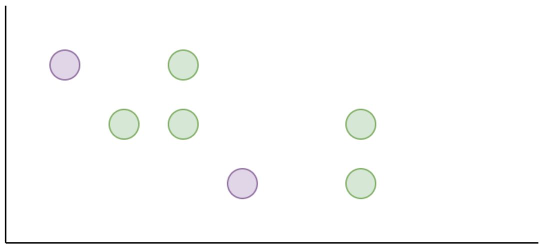

Figure 2: Initialisation: Centroids

2. Each data point is allocated to the nearest centroid

   This "nearness" is based on Euclidean Distance, which equates to measuring the distance between 2 points with a ruler (see Fig. 3).

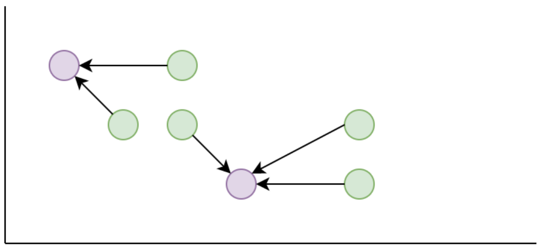

Figure 3: Euclidean distance computed

The result is our clusters. But this is only our first iteration (Fig. 4).

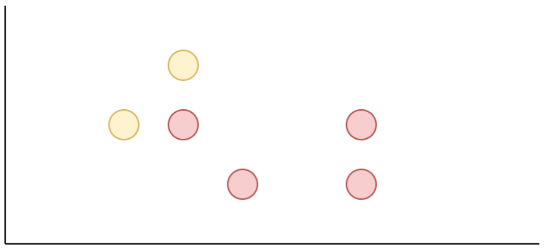

Figure 4: Clusters identified

3. Centroids move to the location of the average of points in their cluster (Fig. 5)

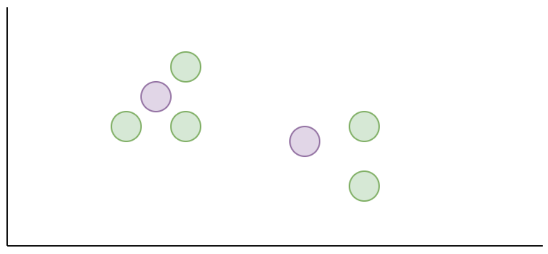

Figure 5: Update for centroids

Then we repeat allocating each point to the nearest centroid as in Fig. 6

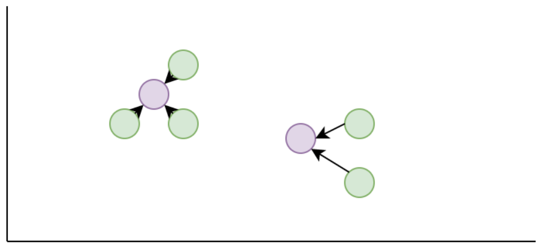

Figure 6: Iteration

This gives us clusters according to our second iteration (see Fig. 7).

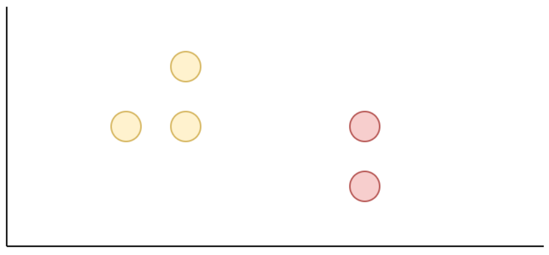

Figure 7: Update for clusters

4. Repeat this process until clusters stop moving

2.3 Formulas

Assuming a point p of dimension n with coordinates p_i, and a point q of dimension n with coordinates q_i then, the euclidean distance $d(p,q)$ is:

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

3 Image segmentation using K-Means

Each pixel of an image is associated to its color described in RGB. The image to be segmented can then be represented as a set of points in a 3D data space.

In case of a grey-level image, the procedure is the same apart from the fact that the image is represented as a set of points in a 1D space.

4 Tasks

1. Overall design: implement relevant classes and their relations (the latter will have to be presented and motivated)
2. Implement a `K-Means` algorithm for a randomly generated dataset for dimension 1, 2, and 3.
   (a) Generate an input cartesian set of points, assuming their dimension, using the implementation of a mersene twister to get real random coordinates.
   (b) Apply the algorithm to this input dataset
3. Implement a `K-Means` algorithm for arbitrary grey level images
   (a) Choose test data images and motivate your choice
   (b) Apply the algorithm to this input dataset as **`tests`**
4. Implement a `K-Means` algorithm for arbitrary color (RGB) images
   (a) Choose test data images and motivate your choice
   (b) Apply the algorithm to this input dataset as **`tests`**

5 Hints

A simple brainstorm, exhibits the following concepts (indentation serves as an indication as well):

- `Input`
  - `GeneratedInput`
  - `ImageInput`
    - `GrayLevelImageInput`
    - `ColorImageInput`
- `Segmentor`
- `Centroid`
- `Cluster`
- `EuclideanRuler`

- `Value`
  - `Pixel` (templated)

These concepts will have to appear in your resulting code, toghether with appropriate relationships.

6 Expectations

6.1 On a scientific level

- How does the choice of K influence the result ?
- How does the choice of the stopping threshold influence the result ?
- To improve the result, we may take into account the color of each pixel together with its position. The color image would then be represented in a 5D space (R,G,B, line, colomn). What would this implementation imply ?

6.2 On a technical level

The resulting program will have to materialise the following elements of the lecture:

- Static Polymorphism
- Dynamic Polymorphism
- Inheritance
- Composition
- Virtual methods
- Templates (usage)
- Operators (whenever relevant)
- Design patterns
  - Singleton
  - Template Methods
- Plainly use the `C++11` standard (including initializers, apply good practices)
- Use the STL (whenever relevant)
- Provide comments with that regard

7 Snippets

1. Find `OpenCV` package on your system

```
find_package(OpenCV REQUIRED)

if(OpenCV_FOUND)
include_directories(${OpenCV_INCLUDE_DIRS})
endif(OpenCV_FOUND)

target_link_libraries(${PROJECT_NAME} PUBLIC ${OpenCV_LIBS})
```

1. Use `OpenCV` to read an RGB image

```
#include <opencv2/core.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui.hpp>
#include <iostream>

using namespace cv;

int main()
{
    std::string image_path = "???.jpg";

    Mat img = imread(image_path, IMREAD_COLOR);

    return 0;
}
```

1. Use `OpenCV` to access pixels

```

cv::Mat img = cv::imread("???.png");

for(int i=0; i<img.rows; i++)
    for(int j=0; j<img.cols; j++)
        // You can now access the pixel value with cv::Vec3b
        std::cout << img.at<cv::Vec3b>(i,j)[0] << " " << img.at<cv::Vec3b>(i,j)[1] << " "
```

8 Datasets

- Generated datasets will have to be presented together with the generation method on the day of the defense
- Grey level images will be provided on the day of the defense to test the program
- RGB images will be provided on the day of the defense to test the program

9 Evaluation modality

- Try and divide the group into subgroups, for each of which you will:
  - define as specific task in the gitlab issue tracker WRT/ milestones
  - segment into subtasks
  - keep and comment the evolutions of the progress for these very (sub)tasks using `Markdown`
- Even though the project is global, the contribution of each participant will be closely followed on your source code repository.
- In order to adapt to any evolution of the implementation, we may change elements of the subject along the way

10 References

- **Singleton Pattern**: `https://sourcemaking.com/design_patterns/singleton`
- **Template Method Pattern**: `https://sourcemaking.com/design_patterns/template_method`

11 Colophon

Emacs 26.3 (Org mode 9.4)