Assignment 4

Basic Image Processing
Fall 2018

History

The algorithm originates from the Mean Shift Clustering method, which is a non-parametric iterative clustering technique introduced in 1975 by Fukunaga and Hostetler.

The idea of “Mean Shift” (i.e. the nonparametric density gradient estimation using a generalized kernel approach) can also be found in many other image processing algorithms: segmentation, visual tracking, space analysis, mode seeking etc.

In this assignment we will implement an image segmentation algorithm based on the Mean Shift idea, using Mean Shift Filtering. This algorithm first does a Mean Shift Filtering and then a Cluster Grouping.

Topological background of Mean Shift

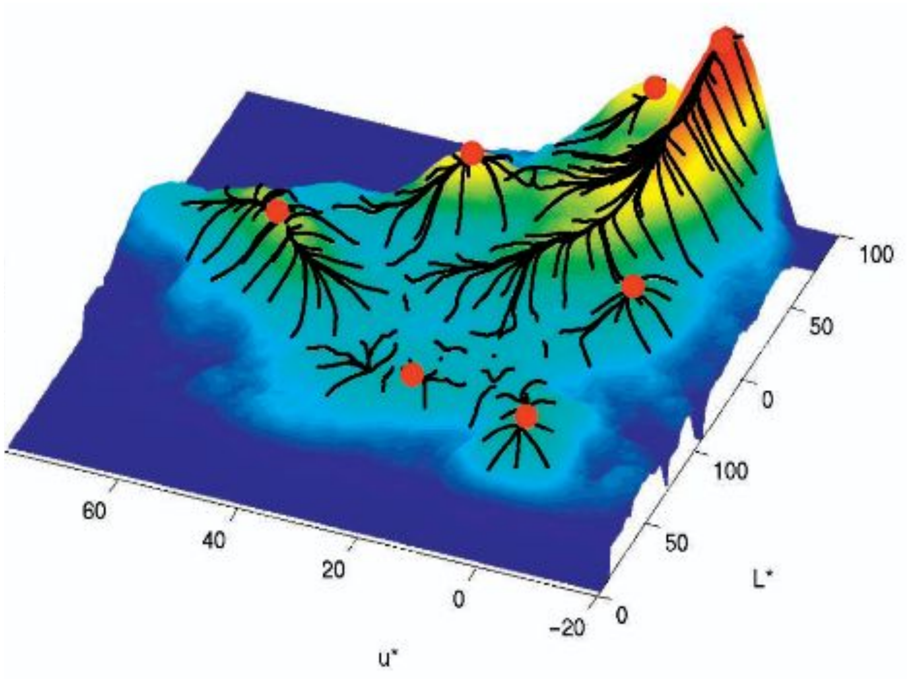
Let F be the YCbCr color space which we call *feature space*.

In this F space every data point is a (1×3) size vector (feature vector) containing `[Y, Cb, Cr]` coordinates alongside the 2nd dimension.

We assume that there is a random variable with an unknown probability density function, and each data point in F is a sample of that random variable.

Topological background of Mean Shift

F

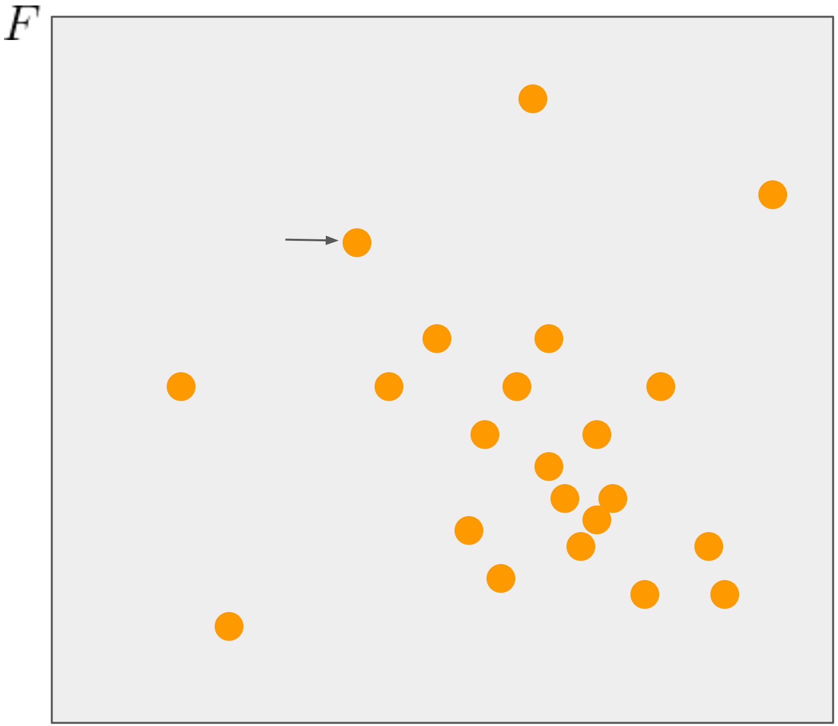


We are looking for the nearest local maximum on the probability density function that we can reach from that data point.

Each data point from which the algorithm leads to the same maximum will belong to the same cluster. Therefore we can clusterize the image based on the different properties of the data points in the F space.

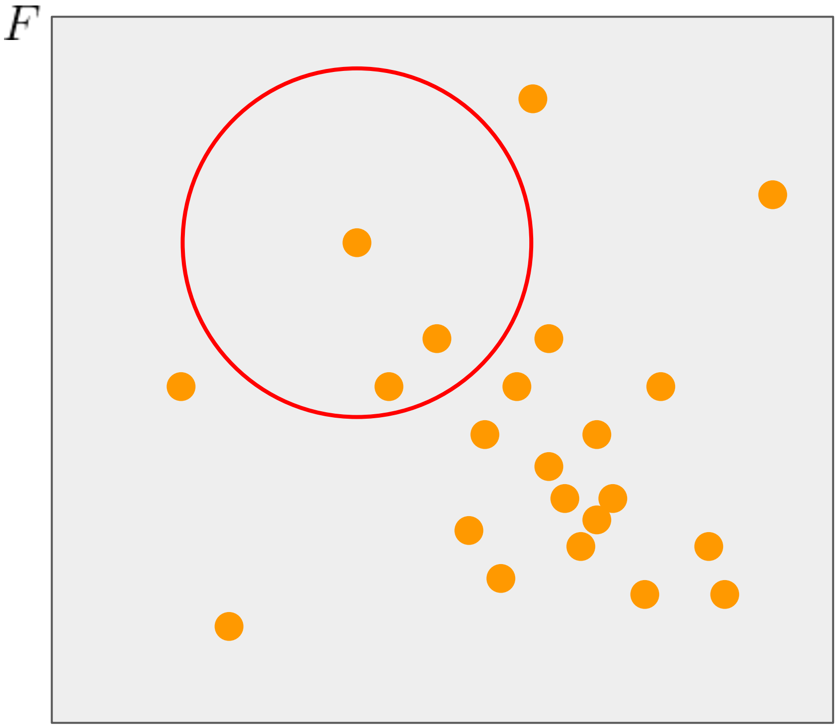
Figure source: https://www.cse.unr.edu/~bebis/CS773C/ObjectRecognition/Papers/Comaniciu02.pdf#page=9

The Mean Shift concept



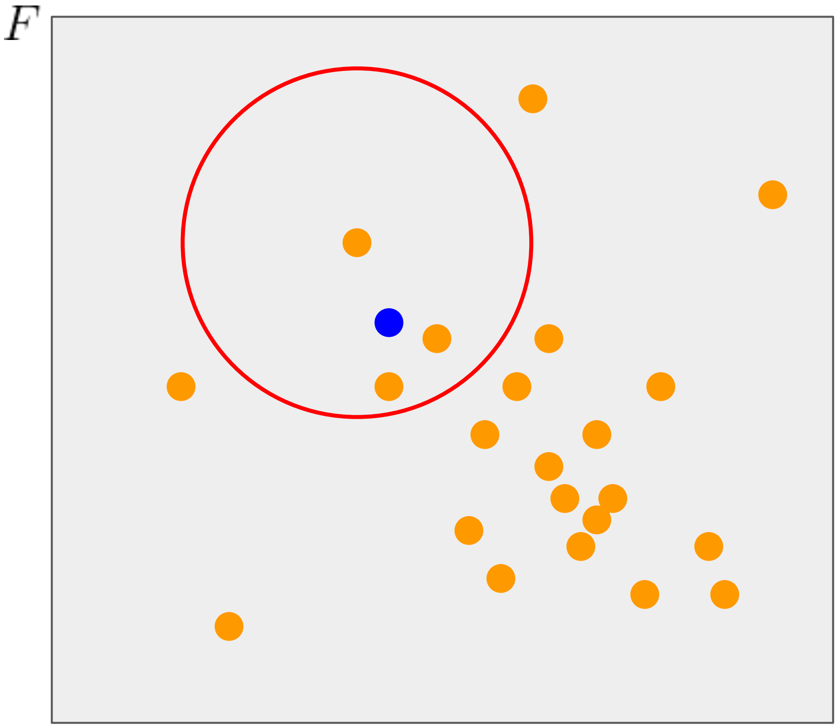
For a data point...

The Mean Shift concept



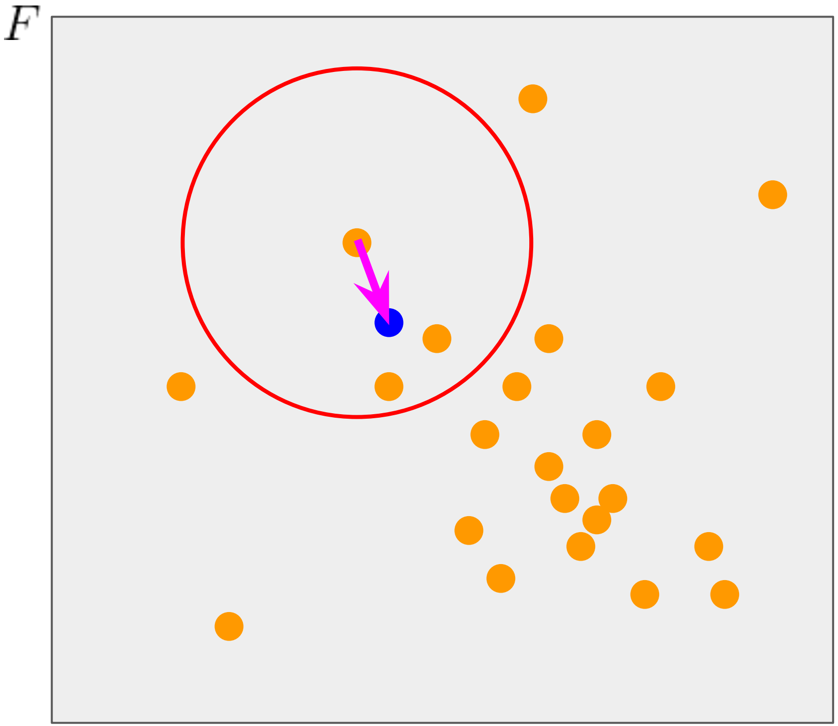
For a data point, mean shift defines a window around it...

The Mean Shift concept



For a data point, mean shift defines a window around it, and computes the mean of data points in the window.

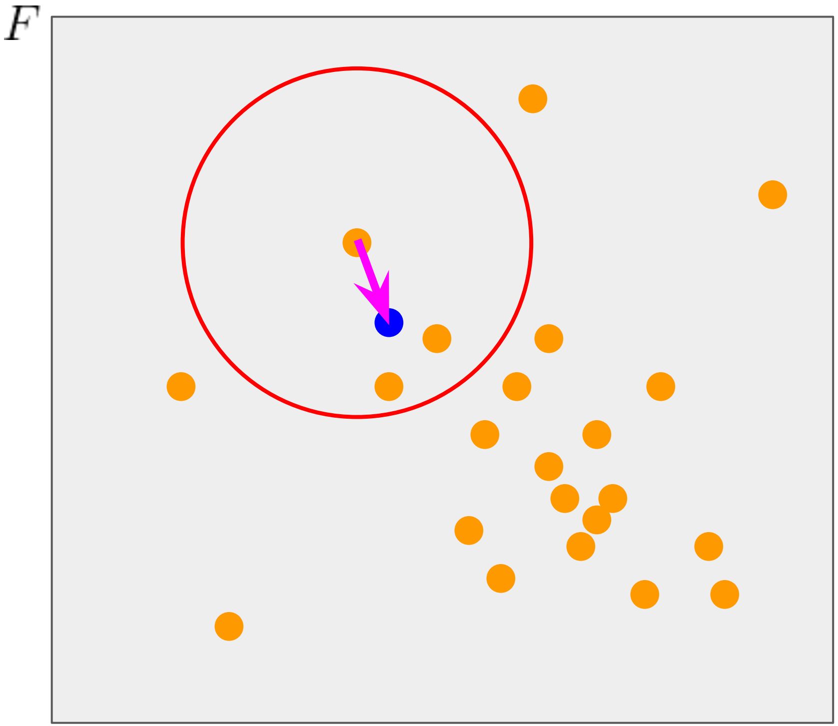
The Mean Shift concept



For a data point, mean shift defines a window around it, and computes the mean of data points in the window.

Then shifts the center of the window to the mean...

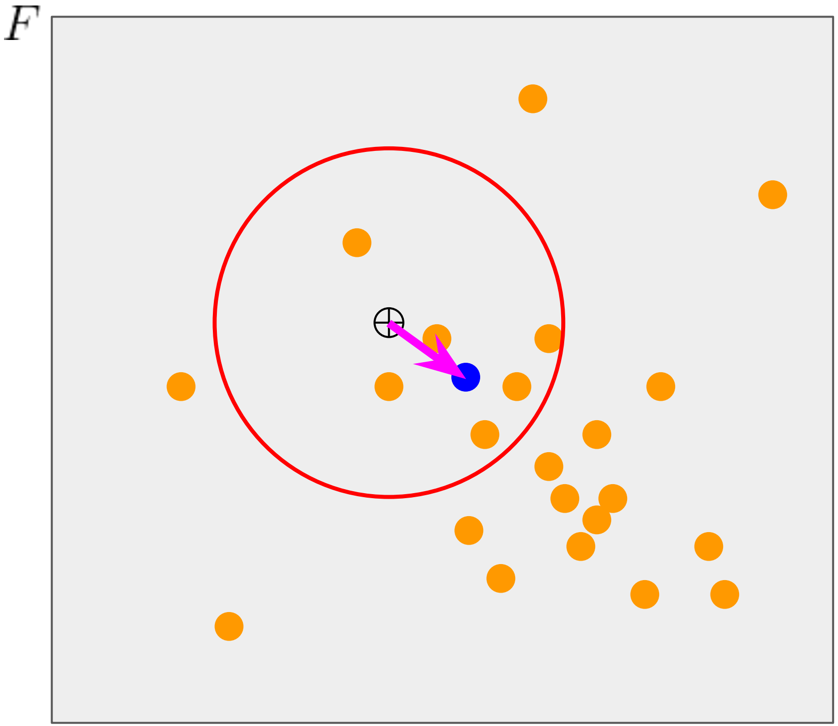
The Mean Shift concept



For a data point, mean shift defines a window around it, and computes the mean of data points in the window.

Then shifts the center of the window to the mean and *repeats* the algorithm...

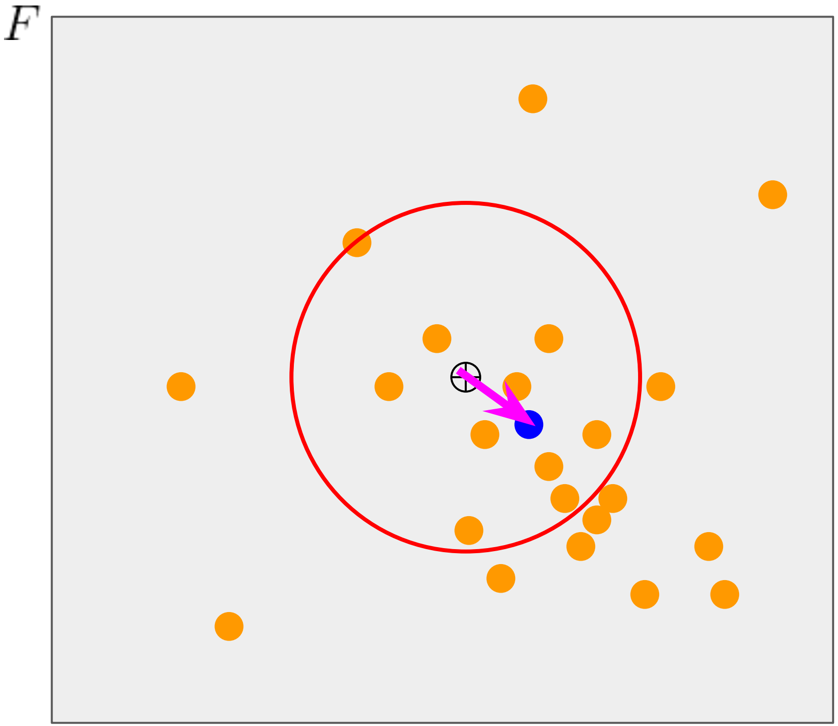
The Mean Shift concept



For a data point, mean shift defines a window around it, and computes the mean of data points in the window.

Then shifts the center of the window to the mean and *repeats* the algorithm...

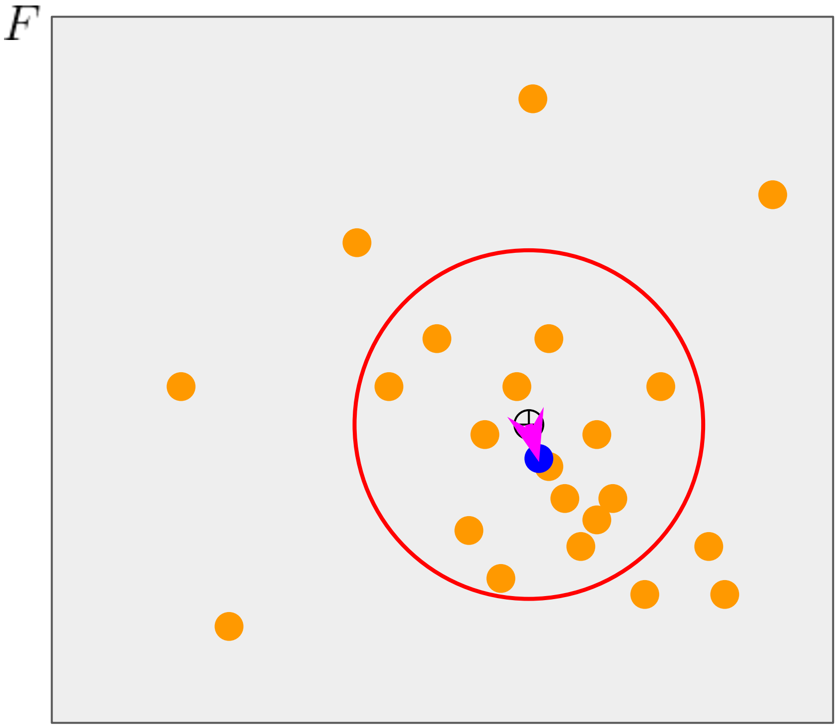
The Mean Shift concept



For a data point, mean shift defines a window around it, and computes the mean of data points in the window.

Then shifts the center of the window to the mean and *repeats* the algorithm...

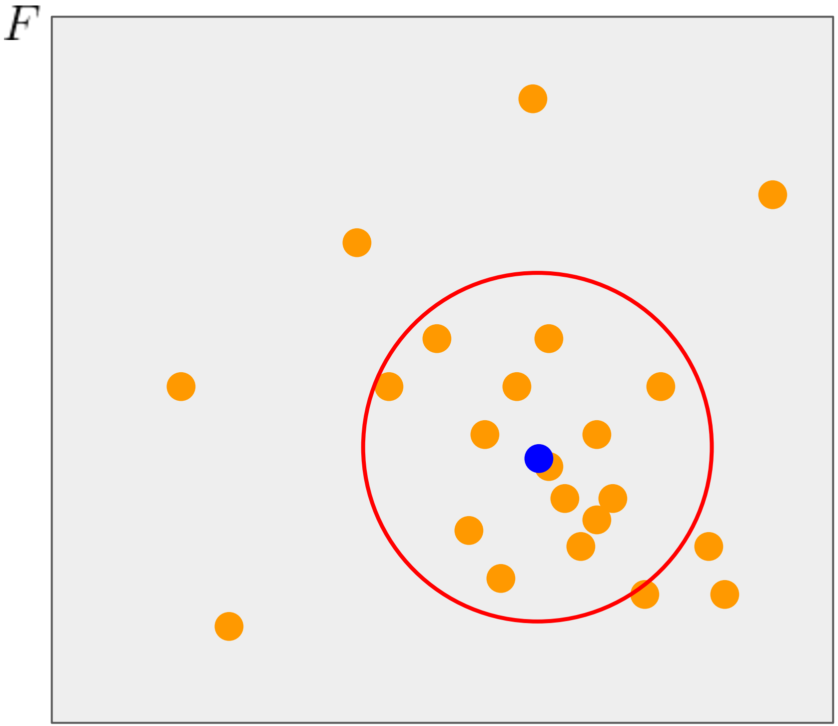
The Mean Shift concept



For a data point, mean shift defines a window around it, and computes the mean of data points in the window.

Then shifts the center of the window to the mean and *repeats* the algorithm...

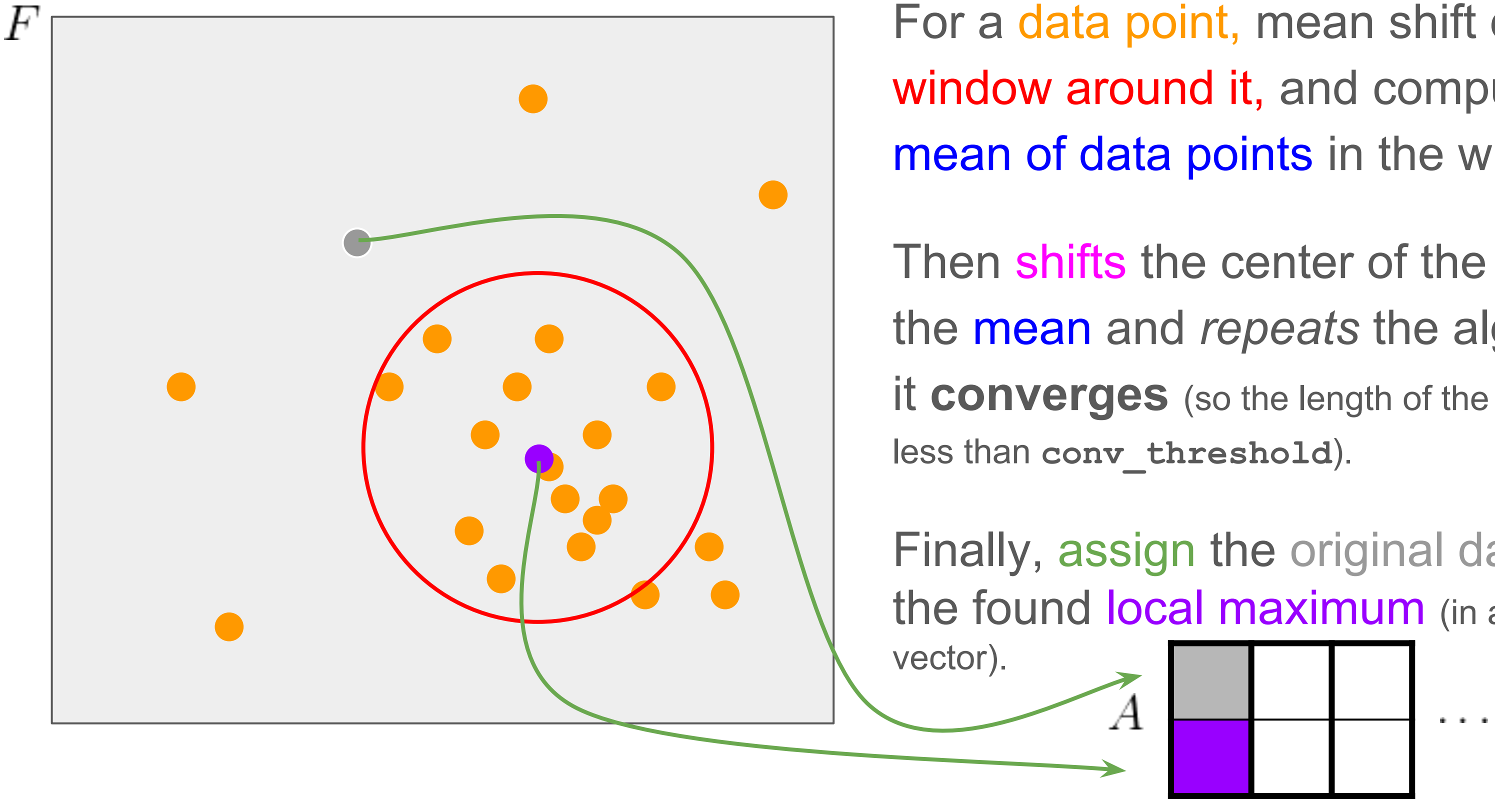
The Mean Shift concept



For a data point, mean shift defines a window around it, and computes the mean of data points in the window.

Then shifts the center of the window to the mean and *repeats* the algorithm until it **converges** (so the length of the shift vector is less than **`conv_threshold`**).

The Mean Shift concept



For a data point, mean shift defines a window around it, and computes the mean of data points in the window.

Then shifts the center of the window to the mean and *repeats* the algorithm until it **converges** (so the length of the shift vector is less than `conv_threshold`).

Finally, assign the original data point to the found local maximum (in an assignment vector).

Mean Shift Filtering

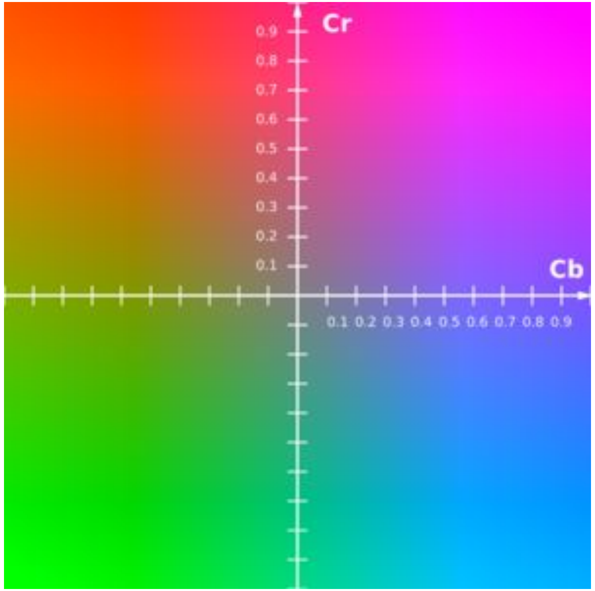
The goal of Mean Shift Filtering is to find the appropriate global maximum for each feature vector of the feature space. This means that for each feature vector we do mean shift steps (as seen on the previous slides) and after convergence we store the endpoints of the trajectory (i.e. the index of the original feature vector and the coordinates of the found maximum point).

The complete Mean Shift Filtering algorithm has the following steps:

- Build the feature space using the RGB image as input;
- For each feature vector, compute the maximum points and store them in a look-up-table (LUT);
- Build the filtered image using the values in the LUT.

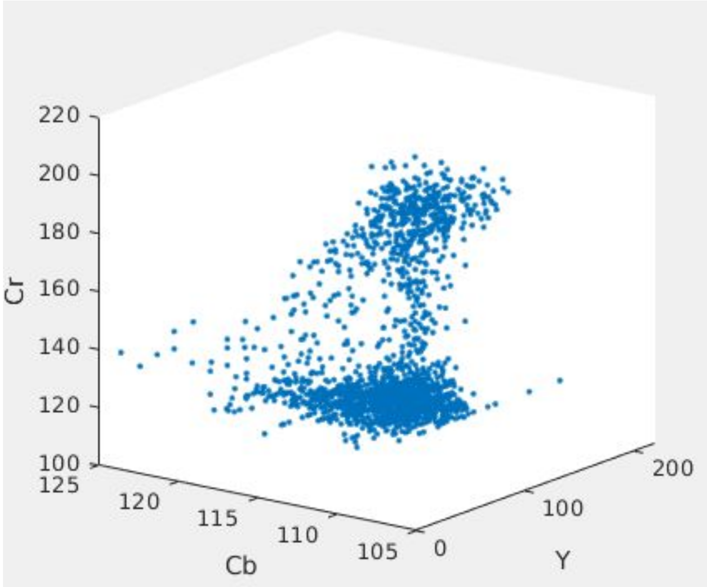
Mean Shift Filtering – building the feature space

We are going to use the *YCbCr* color space as feature space. In this space Y is the luma component, Cb and Cr are chroma components.



The Cb Cr plane at Y = 0.5

The test image



The feature vectors of the test image.

Mean Shift Filtering – computing the shift vector

Let's pick a data point from the feature space and call it $\mathbf{x}$. Then we consider the $N(\mathbf{x})$ neighborhood of this data point, which is defined as a h radius sphere centered in $\mathbf{x}$ (where h is the bandwidth parameter).

The weighted mean in this neighborhood is $\mathbf{m}(\mathbf{x}) = \dfrac{\sum\limits_{\mathbf{x}_i \in N(\mathbf{x})} \mathbf{x}_i K\left(\left\| \dfrac{\mathbf{x} - \mathbf{x}_i}{h} \right\|^2\right)}{\sum\limits_{\mathbf{x}_i \in N(\mathbf{x})} K\left(\left\| \dfrac{\mathbf{x} - \mathbf{x}_i}{h} \right\|^2\right)}$

where K is a (derivative of a) kernel function.

The mean shift vector itself is the vector $\mathbf{m(x)} - \mathbf{x}$

Mean Shift Filtering – iterative approach to find modes

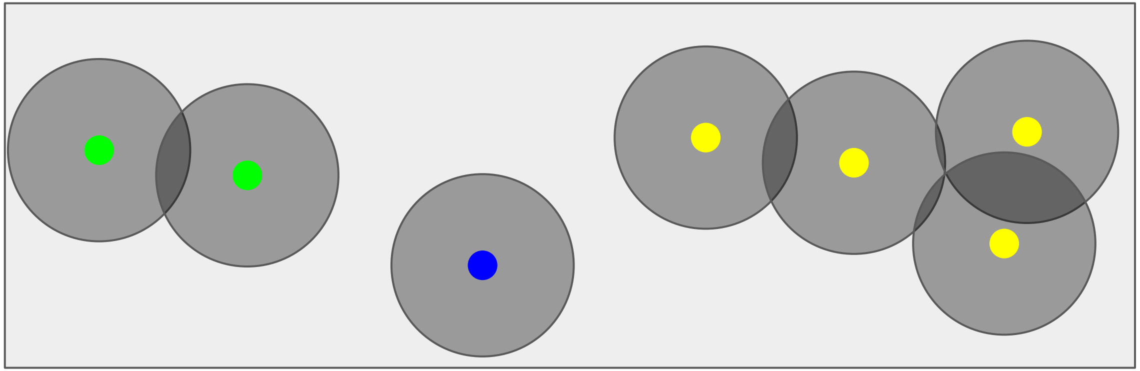
To find the mode (the maximum point where the trajectory ends) for a feature vector we use the following algorithm:

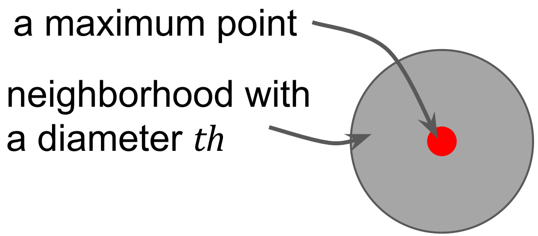
1.) Let's pick an $\mathbf{x}$ feature vector. (i.e. its index in the feature array is given)
2.) While not convergent…
    a.) Select the feature vectors in the h radius neighborhood of $\mathbf{x}$
    b.) Compute their weighted mean (use the kernel function)
    c.) Calculate the shift vector from $\mathbf{m}$ and $\mathbf{x}$
    d.) Let the new $\mathbf{x}$ be $\mathbf{m}$
    e.) Check whether the maximum is found (the approaching converged), so the vector norm of the shift vector is less than a convergence threshold (if yes, exit while loop; if not, continue)
3.) After convergence, return the found maximum vector.

Cluster Grouping

The goal of Cluster Grouping is to join the similar maxima points of the filtered image. The output of the Cluster Grouping step is a cluster map where every pixel of the filtered image is assigned to a cluster label (and hence the original input image is segmented).

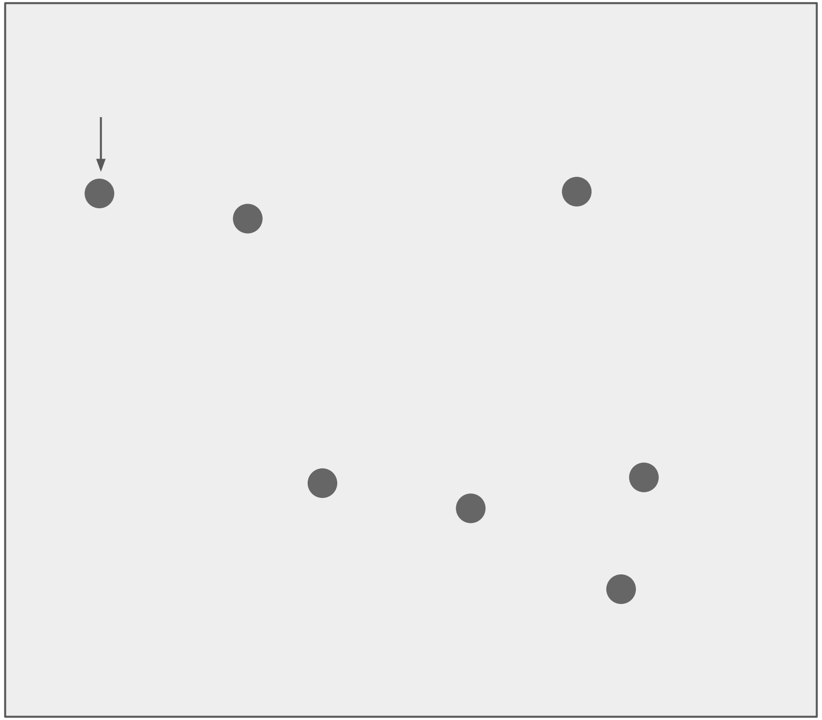
Cluster Grouping uses a th distance threshold parameter: any two maxima points being closer than this threshold will be merged.





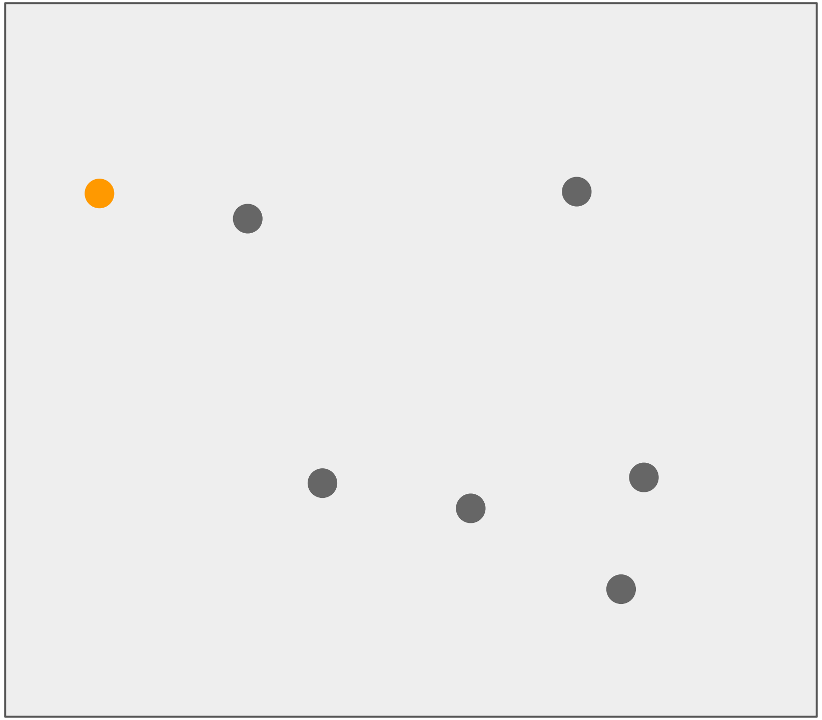
Maxima with overlapping neighborhoods will belong to the same cluster.

Cluster Grouping – the algorithm



First, pick a non-clustered data point.

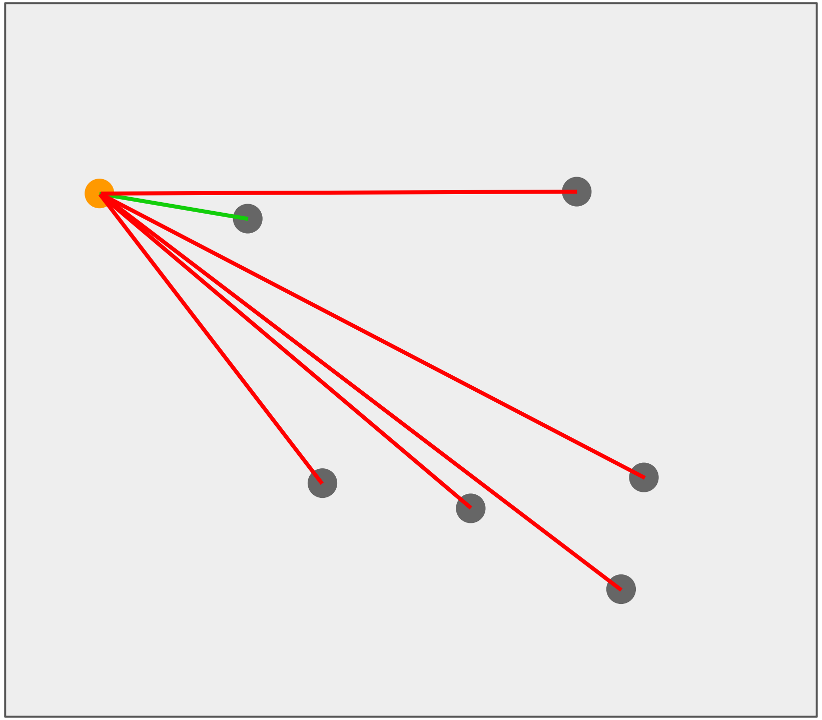
Cluster Grouping – the algorithm



First, pick a non-clustered data point.

Create a new cluster (cluster orange) and label this vector as an element of this cluster.

Cluster Grouping – the algorithm

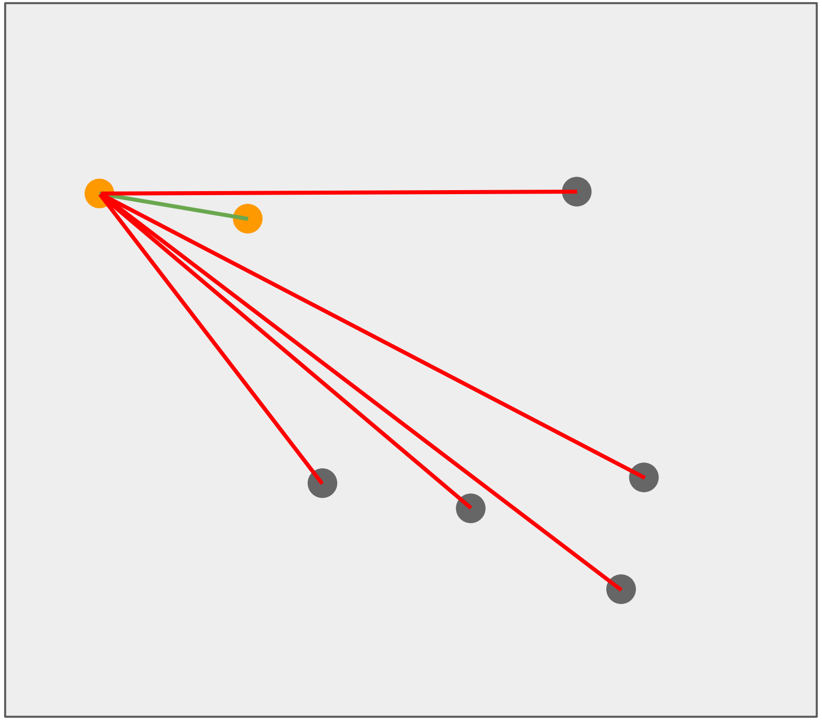


First, pick a non-clustered data point.

Create a new cluster (cluster orange) and label this vector as an element of this cluster.

Compute the Euclidean distances between the chosen data point and all the other, non-clustered points.

Cluster Grouping – the algorithm



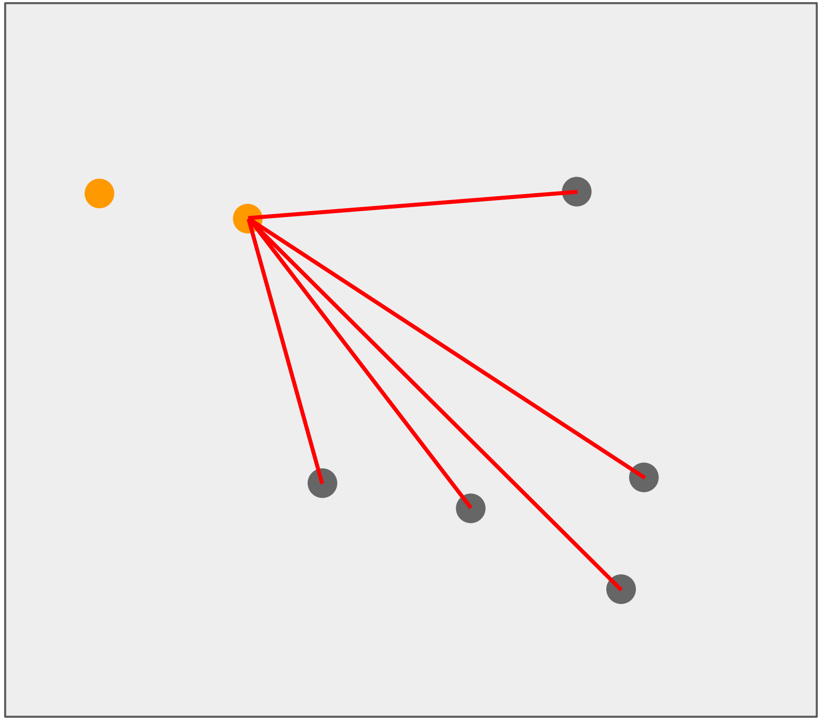
First, pick a non-clustered data point.

Create a new cluster (cluster orange) and label this vector as an element of this cluster.

Compute the Euclidean distances between the chosen data point and all the other, non-clustered points.

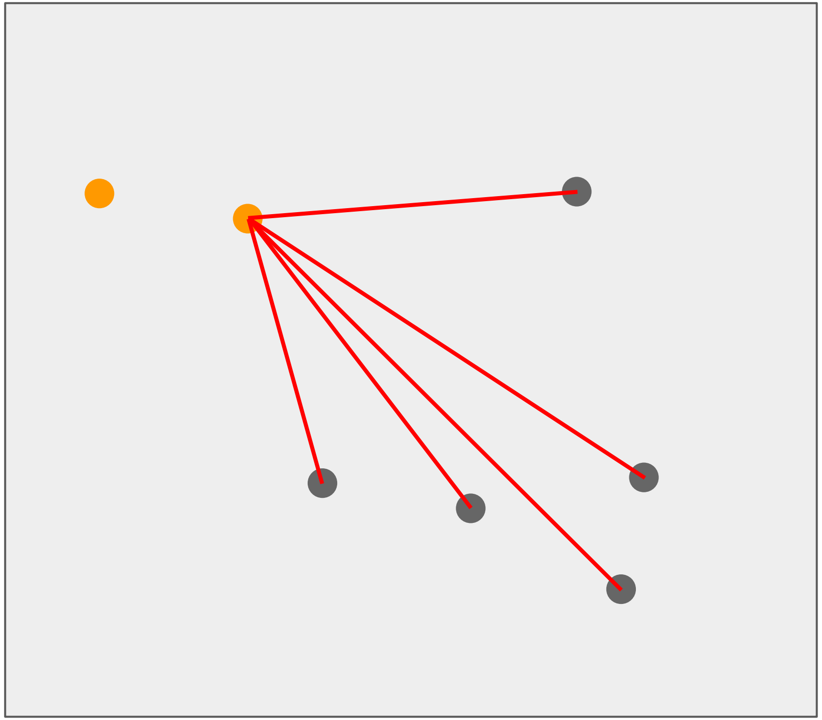
If a point closer than the threshold is found then put it into the same cluster and add it to the queue.

Cluster Grouping – the algorithm



The queue is not empty, so we dequeue its first element and compute the Euclidean distances between the dequeued data point and all the other, non-clustered points.

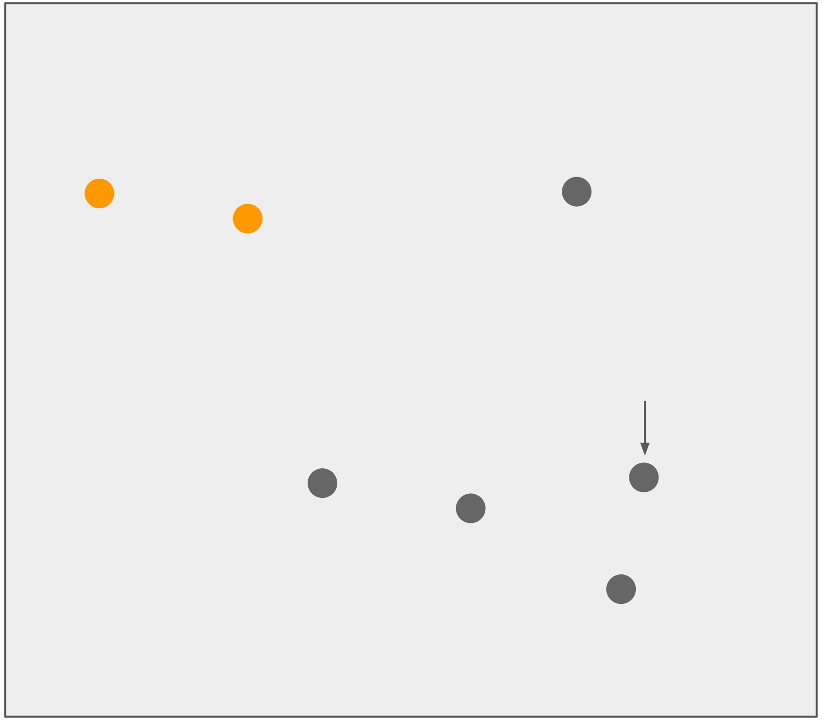
Cluster Grouping – the algorithm



The queue is not empty, so we dequeue its first element and compute the Euclidean distances between the dequeued data point and all the other, non-clustered points.

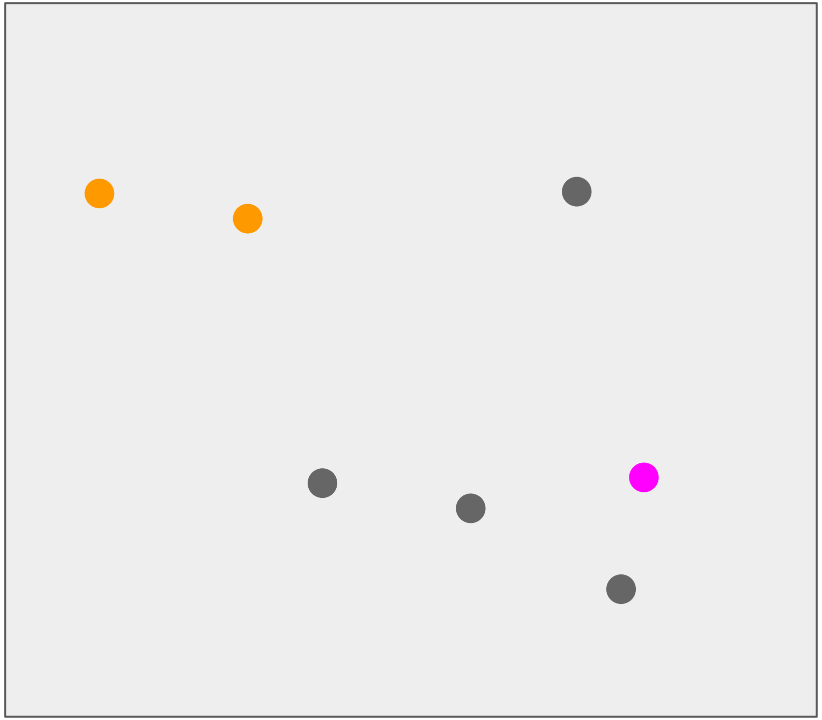
In this example, all the other points are farther from the newly added pont than the threshold. This means that the orange cluster is complete.

Cluster Grouping – the algorithm



To continue the clustering, let’s pick another non-clustered data point.

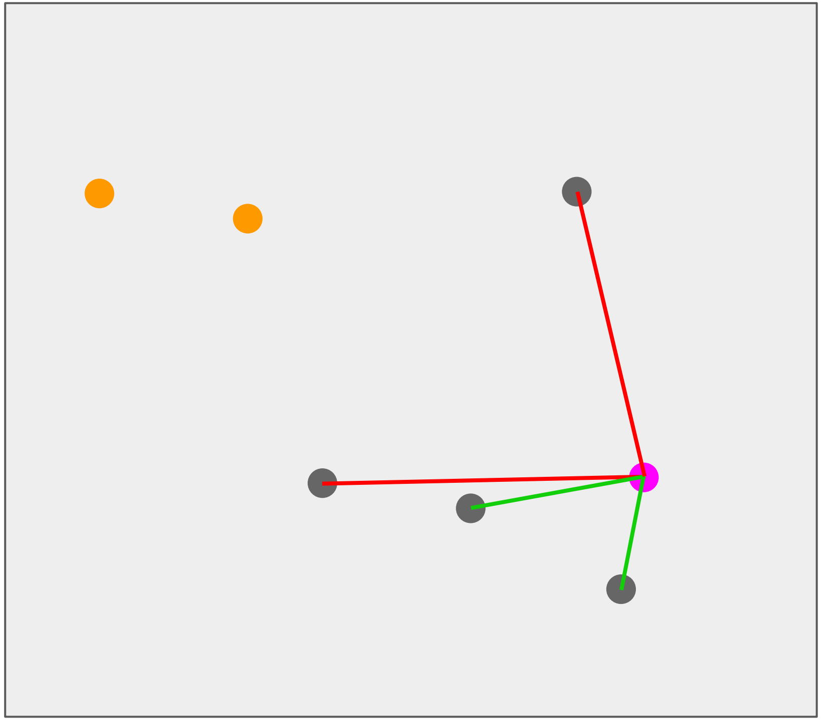
Cluster Grouping – the algorithm



To continue the clustering, let’s pick another non-clustered data point.

Create a new cluster (cluster pink) and label this vector as an element of this cluster.

Cluster Grouping – the algorithm

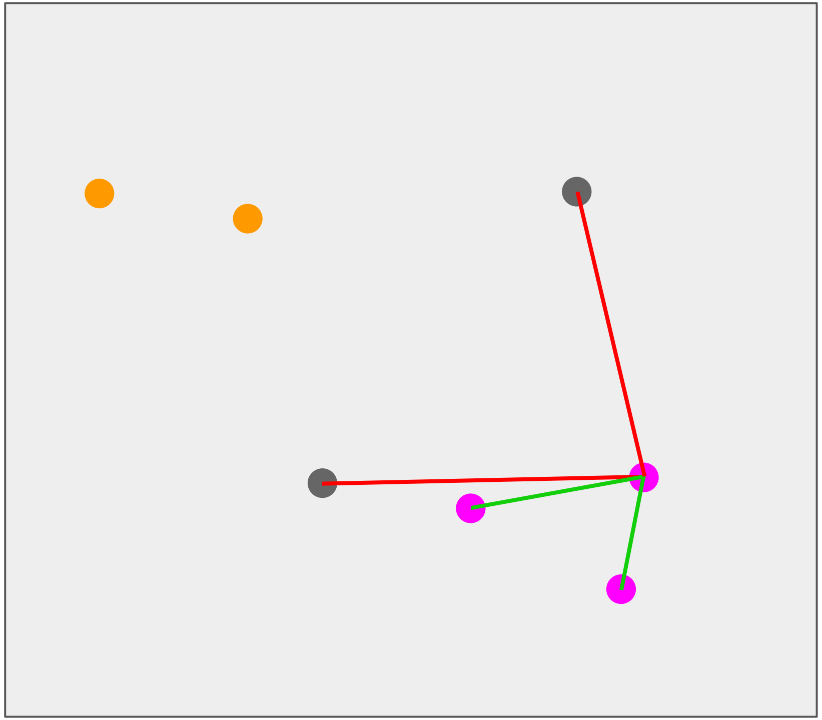


To continue the clustering, let's pick another non-clustered data point.

Create a new cluster (cluster pink) and label this vector as an element of this cluster.

Compute the Euclidean distances between the chosen data point and all the other, non-clustered points.

Cluster Grouping – the algorithm



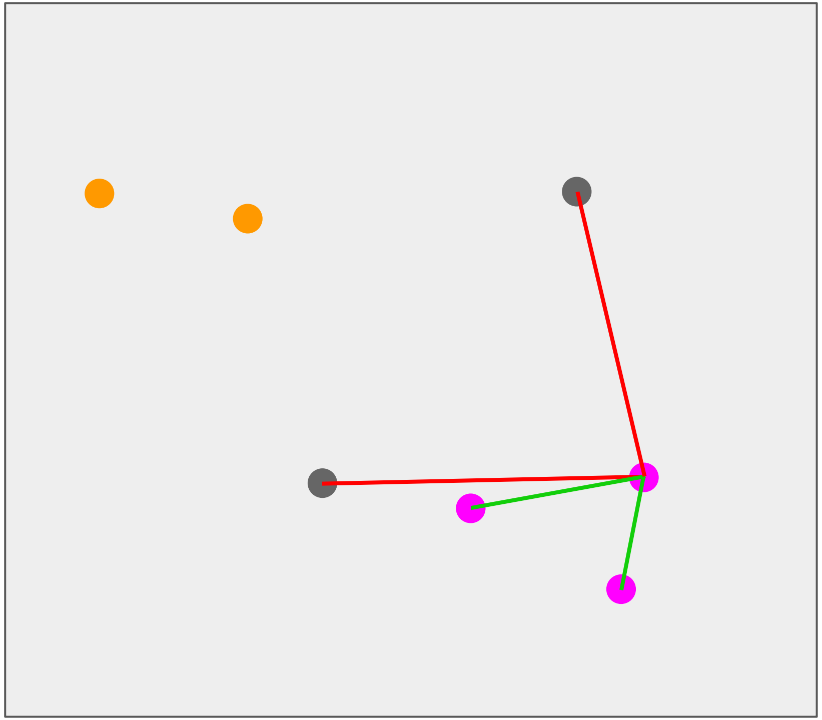
To continue the clustering, let's pick another non-clustered data point.

Create a new cluster (cluster pink) and label this vector as an element of this cluster.

Compute the Euclidean distances between the chosen data point and all the other, non-clustered points.

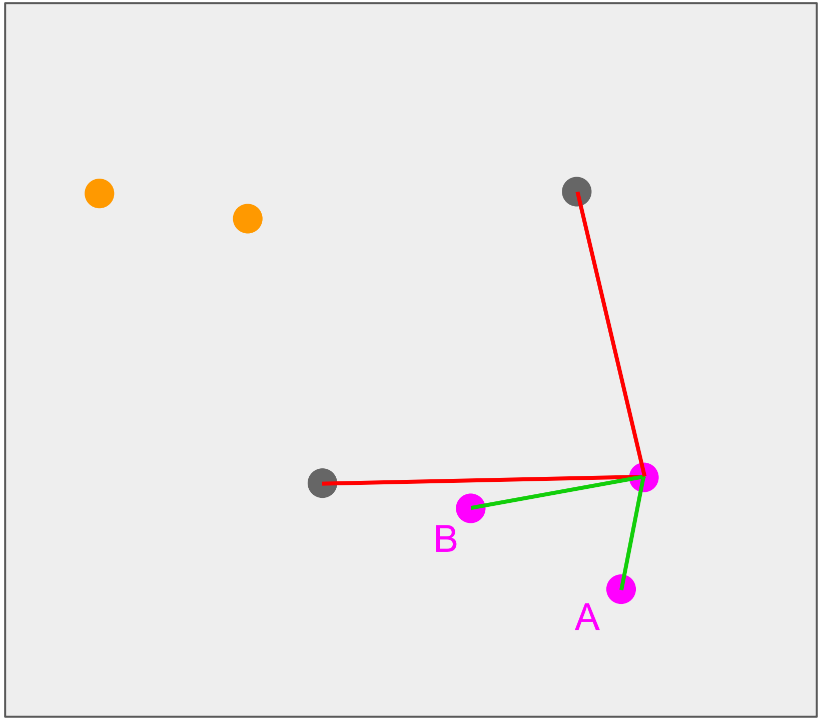
If a point closer than the threshold is found then put it into the same cluster.

Cluster Grouping – the algorithm



In this step, two neighbors were found. We have to compute the distances for both the points. For this, we will use the FIFO queue (like in Assignment 3).

Cluster Grouping – the algorithm

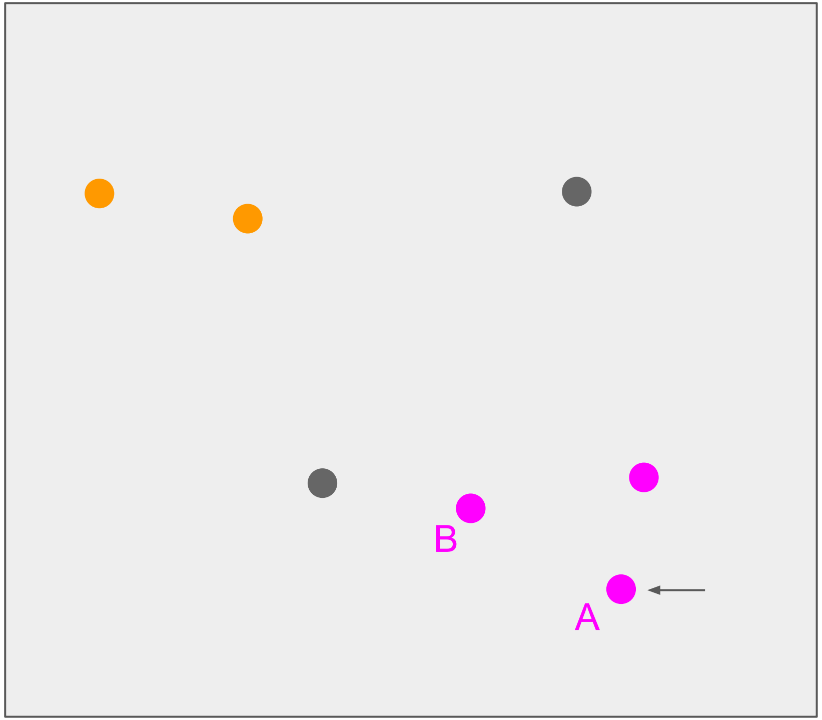


In this step, two neighbors were found. We have to compute the distances for both the points. For this, we will use the FIFO queue (like in Assignment 3).

Put the two neighbor indices into the (currently empty) FIFO:

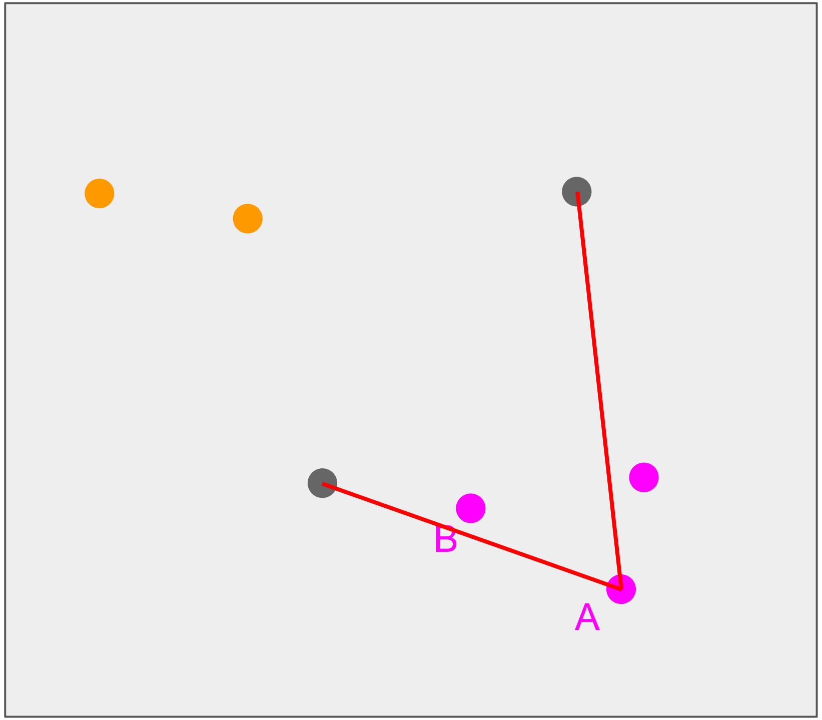
```
HEAD [A, B] TAIL
```

Cluster Grouping – the algorithm



While the FIFO is not empty, dequeue the first element. Now this element is A.

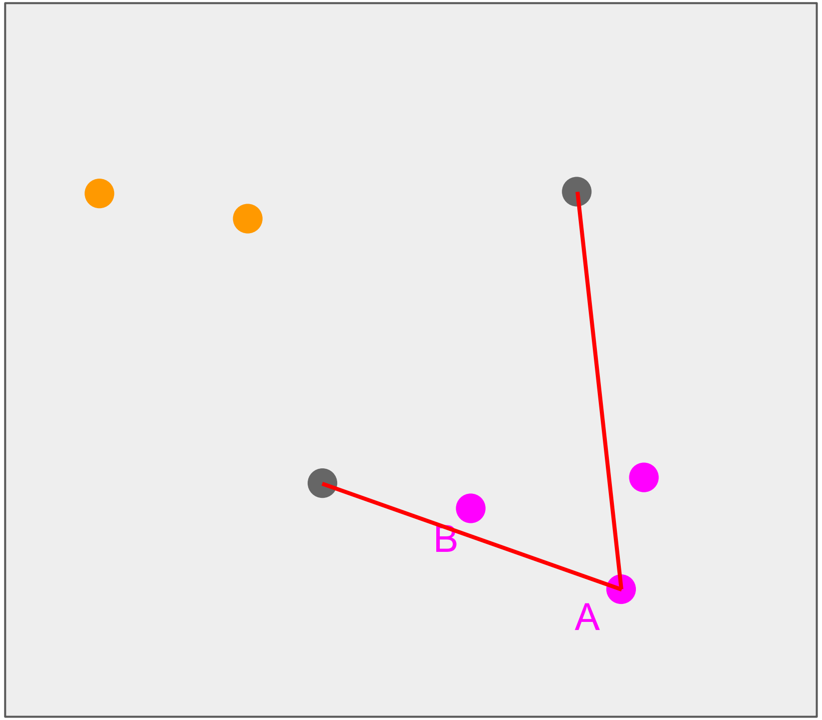
Cluster Grouping – the algorithm



While the FIFO is not empty, dequeue the first element. Now this element is A.

Compute the Euclidean distances between the chosen data point and all the other, non-clustered points.

Cluster Grouping – the algorithm



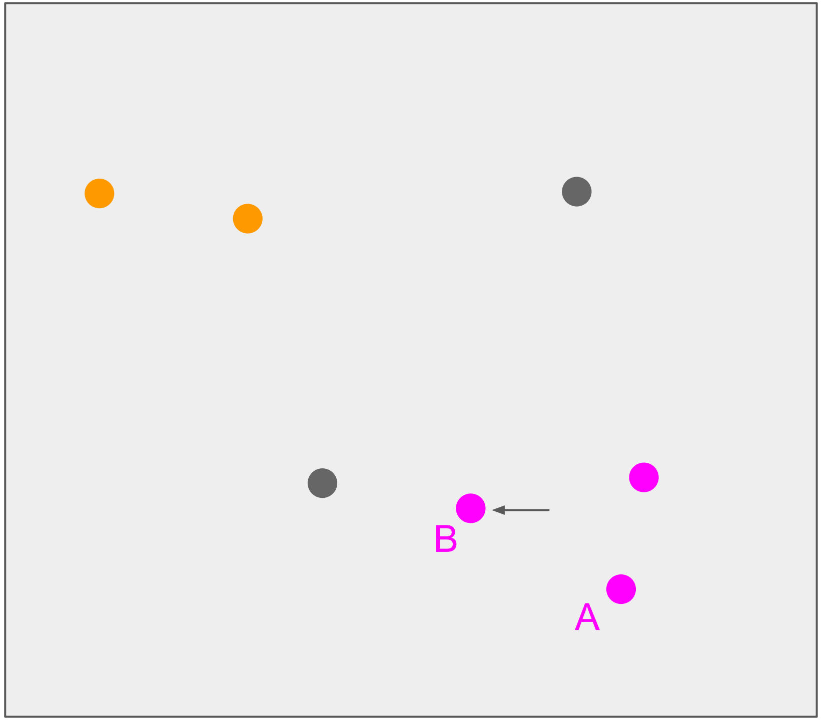
While the FIFO is not empty, dequeue the first element. Now this element is A.

Compute the Euclidean distances between the chosen data point and all the other, non-clustered points.

Since in this example all the distances are larger than the threshold, there is no cluster label assignment and enqueue step.

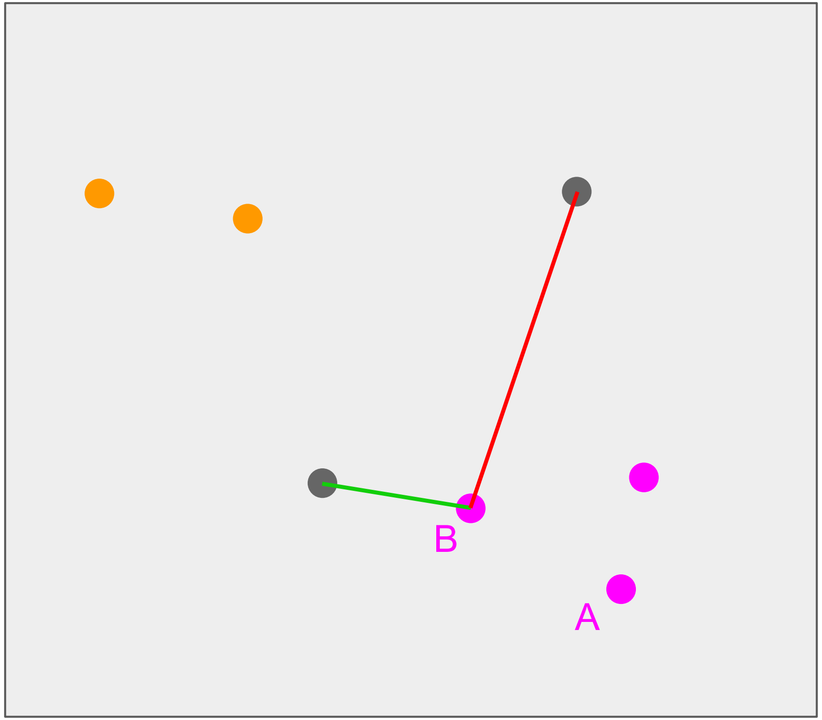
We continue with the next element in the queue.

Cluster Grouping – the algorithm



While the FIFO is not empty, dequeue the first element. Now this element is B.

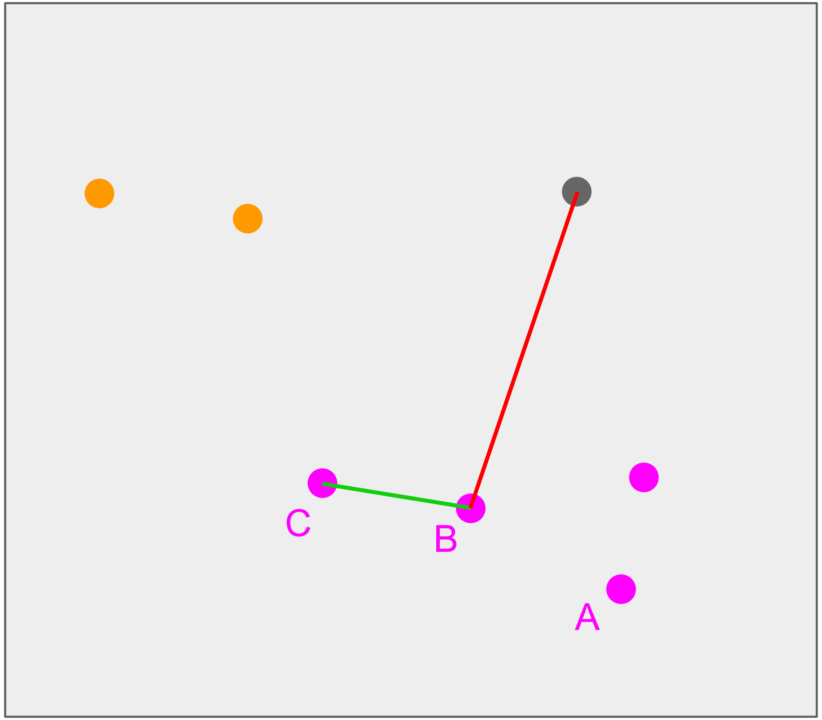
Cluster Grouping – the algorithm



While the FIFO is not empty, dequeue the first element. Now this element is B.

Compute the Euclidean distances between the chosen data point and all the other, non-clustered points.

Cluster Grouping – the algorithm



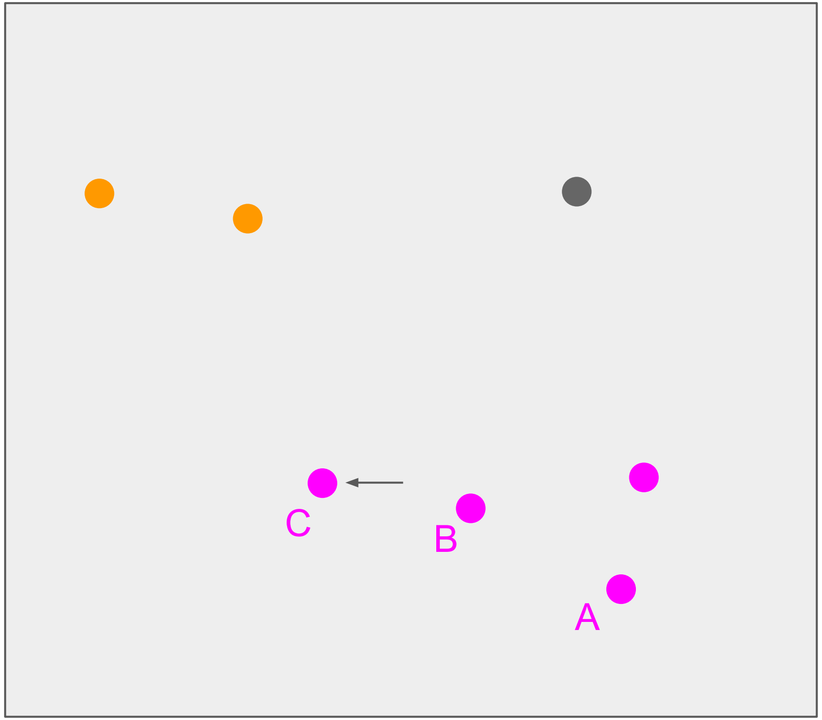
While the FIFO is not empty, dequeue the first element. Now this element is B.

Compute the Euclidean distances between the chosen data point and all the other, non-clustered points.

There is a distance which is smaller than the threshold, so we put this element in the same cluster and add it to the FIFO as well. Now the FIFO contains only one element:

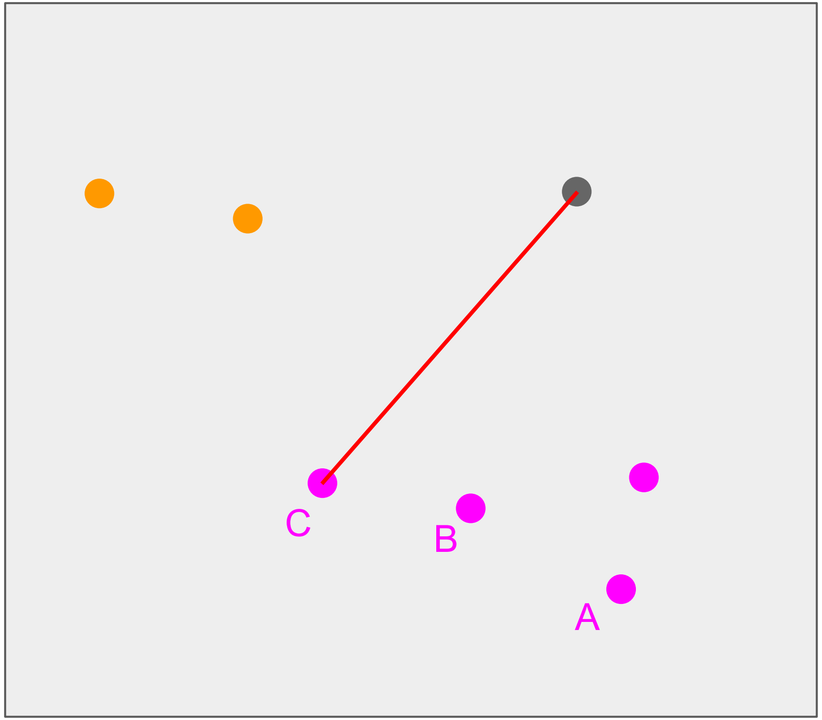
```
HEAD [C] TAIL
```

Cluster Grouping – the algorithm



While the FIFO is not empty, dequeue the first element. Now this element is C.

Cluster Grouping – the algorithm

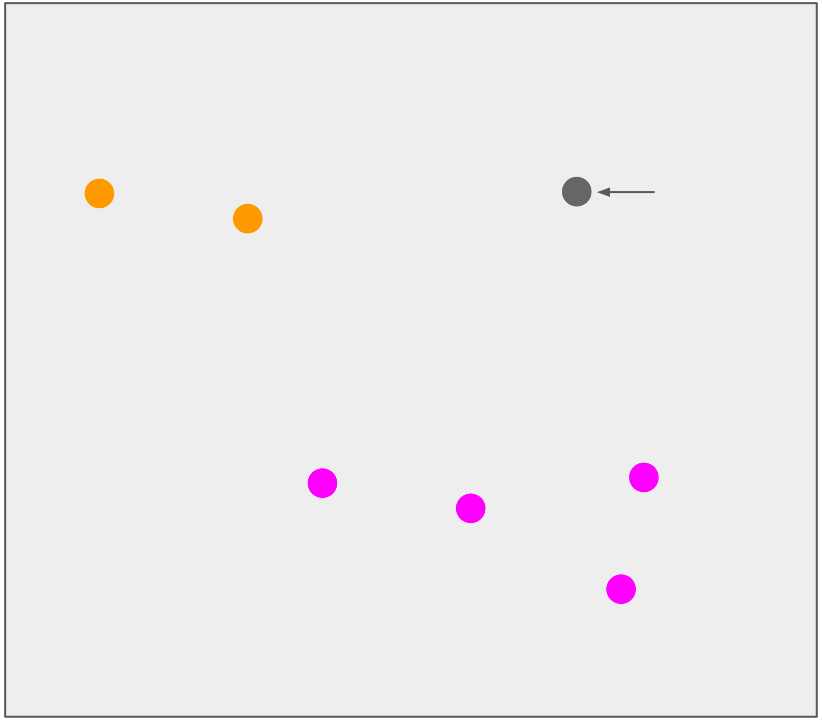


While the FIFO is not empty, dequeue the first element. Now this element is C.

Compute the Euclidean distances between the chosen data point and all the other, non-clustered points.

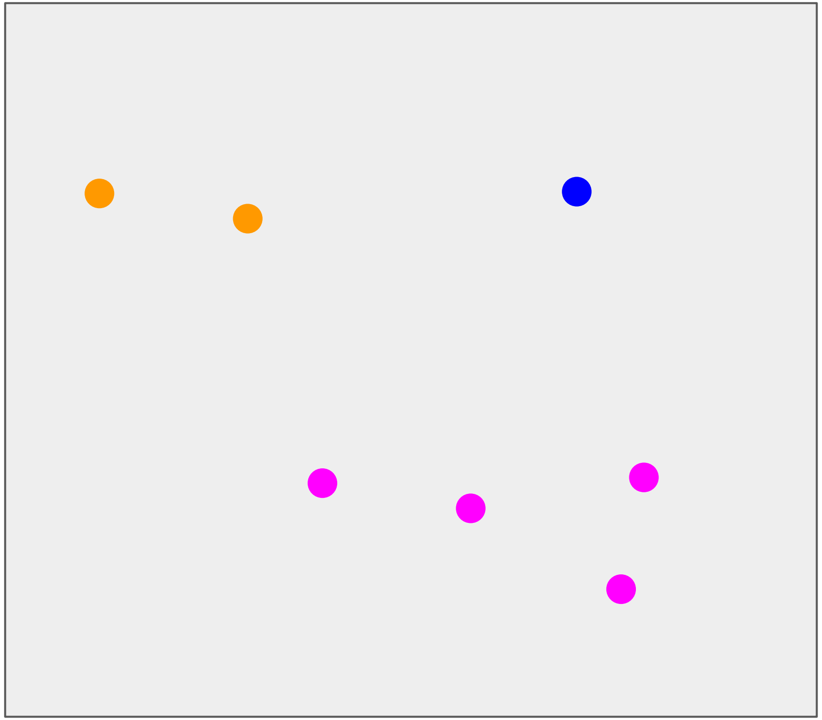
Since in this example this distance is larger than the threshold, there is no cluster label assignment and enqueue step. Also, as the FIFO is empty now, cluster pink is complete and in the next step we will create a new cluster.

Cluster Grouping – the algorithm



To continue the clustering, let's pick another non-clustered data point.

Cluster Grouping – the algorithm



To continue the clustering, let's pick another non-clustered data point.

Create a new cluster (cluster blue) and label this vector as an element of this cluster.

There are no more non-clustered data points so the algorithm is finished.

Please

download the ‘Assignment 4’ code package

from the

[submission system]

The maximum score of this assignment is

7 points

The points will be given in 0.25 point units.
(Meaning that you can get 0, 0.25, 0.5, 0.75, 1, 1.25 etc. points).

Exercise 1

Implement the function `feature_extractor` in which:

- The input is the uint8 type RGB `image_matrix` as a 3D array
- The output is the double type `feature_space` as a 2D array: every row is a feature vector describing a pixel.

Use the function `rgb2ycbcr()` to transform the RGB image to the YCbCr color space. Please, `reshape()` the YCbCr image into an $N \times 3$ size array, this will be the feature space. (N is the number of pixels in the original image). Please make sure that the values in the feature space are double type.

You can test this function using `test1.m`.

Exercise 2

Implement the function `kernel_function` in which:

- The input is a number (double) (`x`)
- The output is the response of the derivative of the ‘normal’ kernel.

The derivative of the ‘normal’ kernel is described as the following:

$$K(\mathbf{x}) = \frac{1}{2} e^{-\frac{1}{2}\mathbf{x}}$$

You can test this function using `test2.m`.

Exercise 3

Implement the function `find_mode` in which:

- The inputs are:
  - the `feature_space`,
  - the `index` of the feature vector used as starting point
  - the `bandwidth` (neighborhood radius), and
  - the convergence threshold (`conv_threshold`).
- The output is the 1×3 vector of the mode (the point to which the starting point converges to).

More details on the next slide!

You can test this function using `test3.m`.

Exercise 3

This function runs a mean shift filtering for one feature vector selected by the index input argument. This function should realize the converging mean shift approach described on Slide 18.

It is **very important** to understand the concept of the neighborhood: **you have to select those feature vectors whose Euclidean distance from the center point is less than the radius given by the bandwidth.** So it can happen that you have to select the 3rd, 5th, 6th and 8th vector from the feature space. Selecting the feature vectors with neighboring *indexes* is **wrong**.

		Euclidean distance	These are the vectors in the neighborhood
F =	[1 0 0]	5.00	
	[1 1 1]	4.12	
	[0 1 5]	1.41	[0 1 5]
	[8 9 2]	11.8	
	[1 0 5]	0.00	[1 0 5]
	[0 2 7]	3.00	[0 2 7]
	[7 0 1]	7.21	
	[1 0 4]	1.00	[1 0 4]

In this example index = 5, so this is the center element

In this example bandwidth = 4, so we select these elements

Exercise 4

Implement the function `mean_shift_filtering` in which:

- The inputs are:
  - the `feature_space`,
  - the `bandwidth` (neighborhood radius), and
  - the convergence threshold (`conv_threshold`).
- The output is the filtered space, having the same size as the feature space.

In this function you have to call the `find_mode` function for every feature vector. Save the found mode in `filtered_space` at the appropriate location.

You can test this function using `test4.m`.

Exercise 5

Implement the function `join_regions` in which:

- The inputs are:
  - the `filtered_space`, and
  - the distance threshold (`distance_th`).
- The output is the cluster space as an $N \times 1$ column vector (where N is the number of feature vectors in the filtered space).

In this function you should visit all the feature vectors. If a distance between two vectors is less than the threshold then you should join the two regions.

More details on the upcoming slides!

You can test this function using `test5.m`.

Exercise 5

Create the cluster space as an $N \times 1$ column vector filled with zeros (N is the number of feature vectors in the filtered space). Initialize the FIFO as an empty cell.

While there are feature vectors without cluster label…

If the FIFO is empty, pick a feature vector without cluster label, create a new cluster, assign its label to the feature vector, compute the distance between this vector and all the other non-clustered ones, select those that are closer than the threshold, assign the cluster label to them and put them in the FIFO.

If the FIFO is not empty, dequeue the first feature vector of FIFO, compute the distance between this vector and all the other non-clustered ones, select those that are closer than the threshold, assign the same cluster label to them and put them in the FIFO.

Exercise 6

Implement the function `reshape_spaces` in which:

- The inputs are:
  - `the_space` to be reshaped, and
  - the size of the original input image (`image_size`).
- The output is a $h \times w \times 3$ or $h \times w$ matrix depending on the size of `the_space`. (The h and w are the height and width of the original image.)

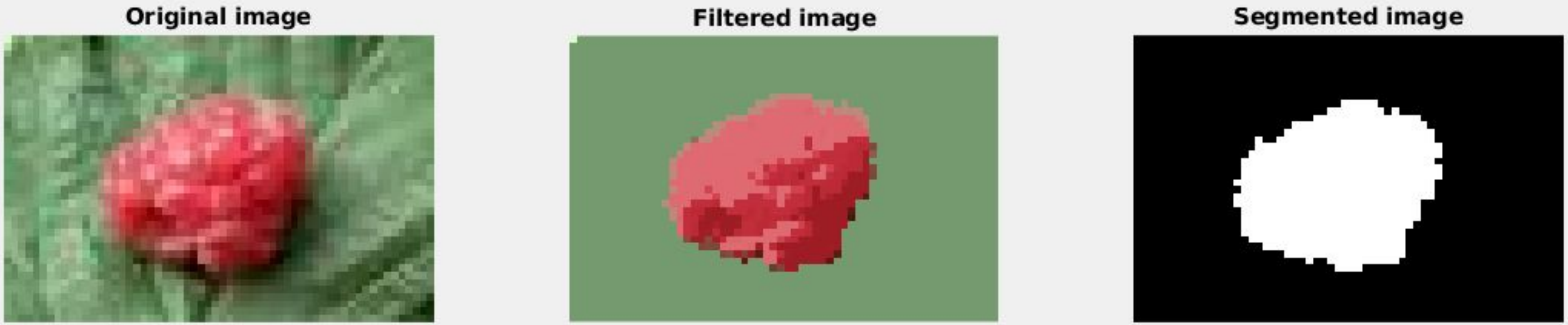
If the input is an $N \times 3$ array (the YCbCr feature space) then the output should be a uint8 RGB image. Use `reshape()`, `ycbcr2rgb()` and `uint8()` functions.

If the input is an $N \times 1$ column vector (the cluster space) then the output should be a uint8 grayscale image. Use `reshape()`, `mat2gray()` and `uint8()` functions.

You can test this function using `test6.m`.

A good result...

If you are ready with all the implementations, run the `do_segmentation` script.
The result should be something **similar** **(your result can be slightly different!)**:




```
Segmentation finished in 4.844032 seconds.
```

THE END