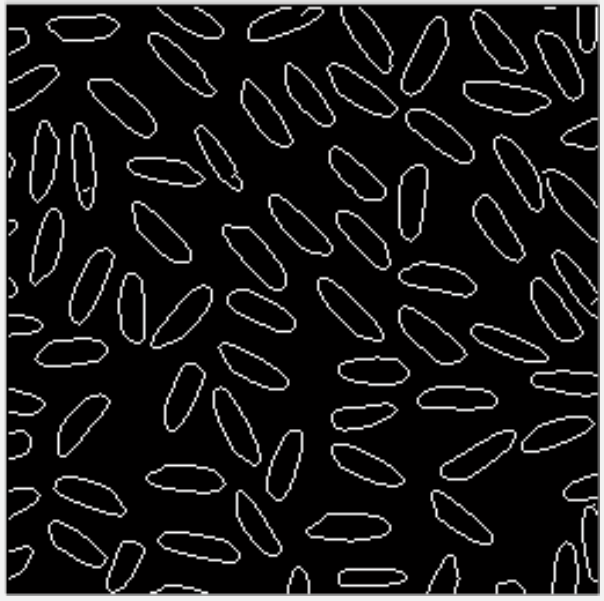
Department of Computer Science
University of Bristol

COMS30121 - Image Processing and Computer Vision

www.ole.bris.ac.uk/bbcswebdav/courses/COMS30121_2017/content



Lecture 05

Segmentation Basics

Andrew Calway | andrew@cs.bris.ac.uk
Tilo Burghardt | tilo@cs.bris.ac.uk
Majid Mirmehdi | majid@cs.bris.ac.uk

30 Slides

Examples of Image Segmentation

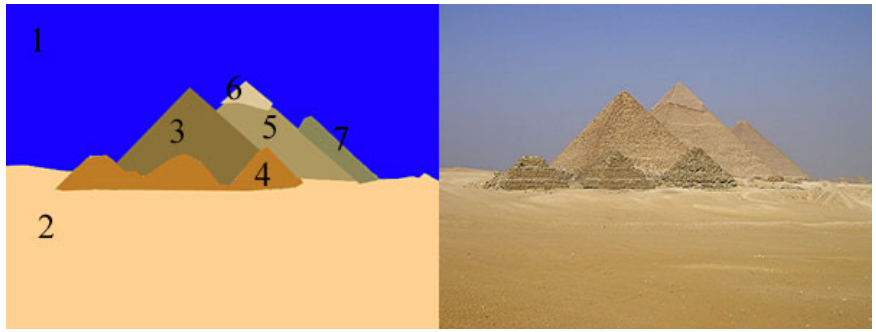
- **Image Segmentation ...**

  ... is the process of spatial subsectioning of a (digital) image into multiple partitions of pixels (i.e. segments or regions) according to given criteria.

Example: segmentation of an image into locally coherent regions

Motivation: Why Segment Images?

- **Image Simplification**
  - an image may contain millions of pixels but only a few regions



- **Higher-level Object Description**
  - regions tend to belong to the same class of object
  - regions may provide object properties (e.g. shape, colour, ...)

- **Input for Content Classifiers**
  - region descriptions can be input data for higher level classifiers, e.g. Bayesian Classifiers or Neural Networks.

Digression: Gestalt-Rules for Grouping and Segmentation

When we look at the world, we usually perceive complex scenes composed of many groups of objects on some background, with the objects themselves consisting of parts, which may be composed of smaller parts, etc. How do we accomplish such a remarkable perceptual achievement, given that the visual input is, in a sense, just a spatial distribution of variously colored individual points? Dejan Todorovic, Scholarpedia

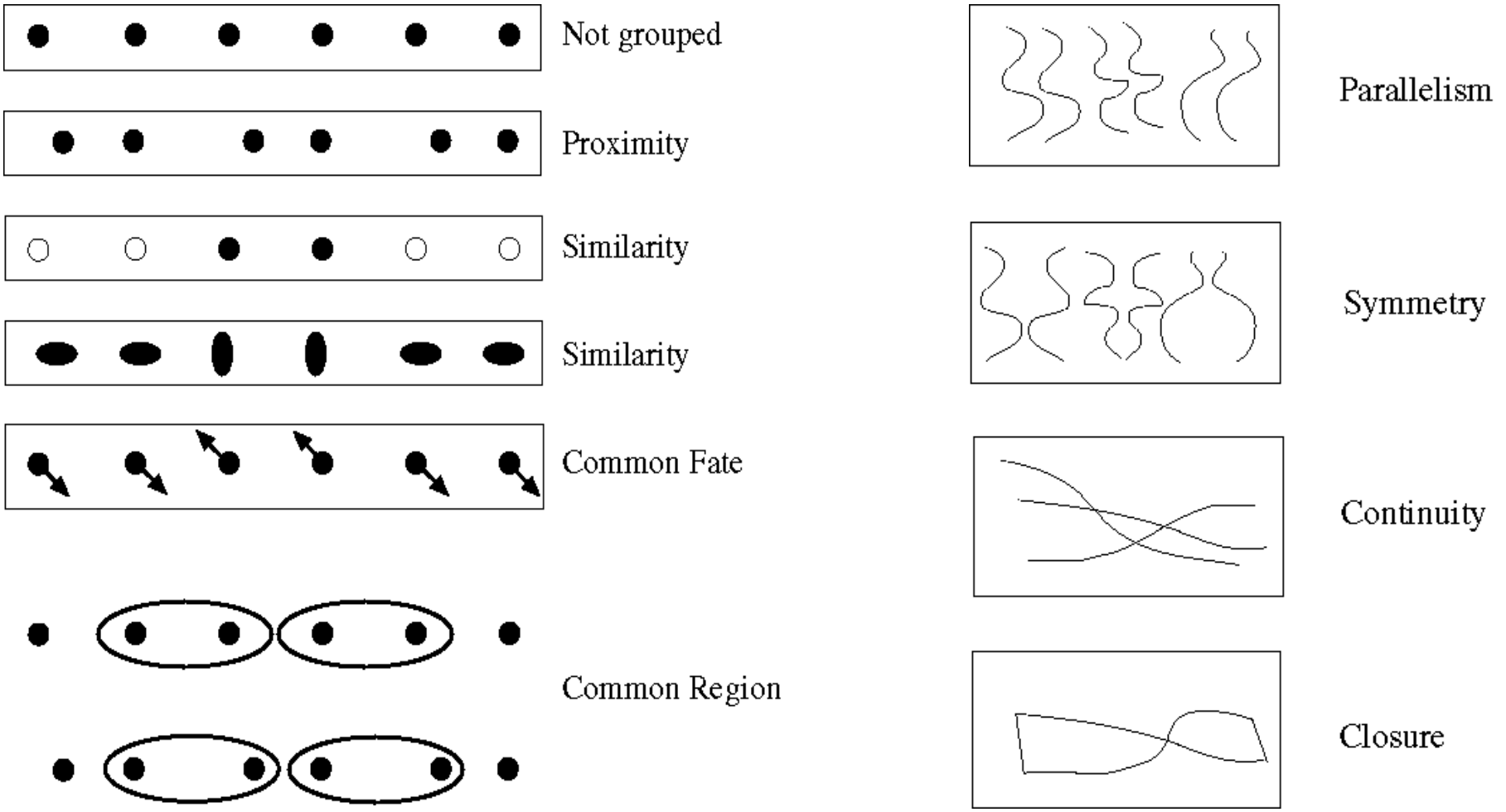
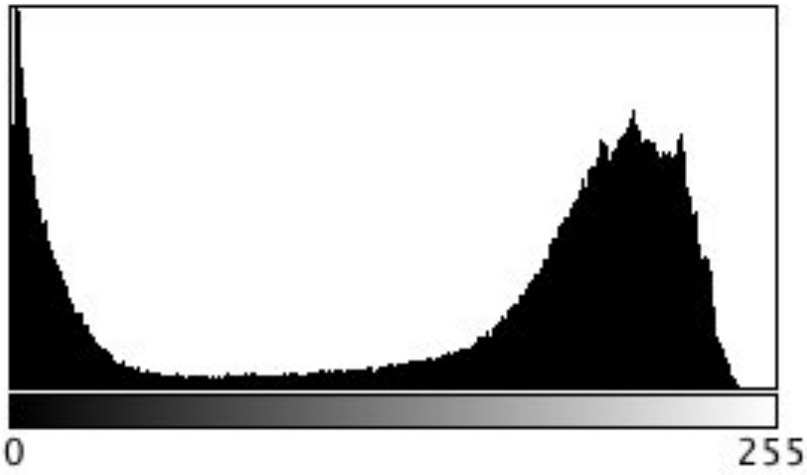
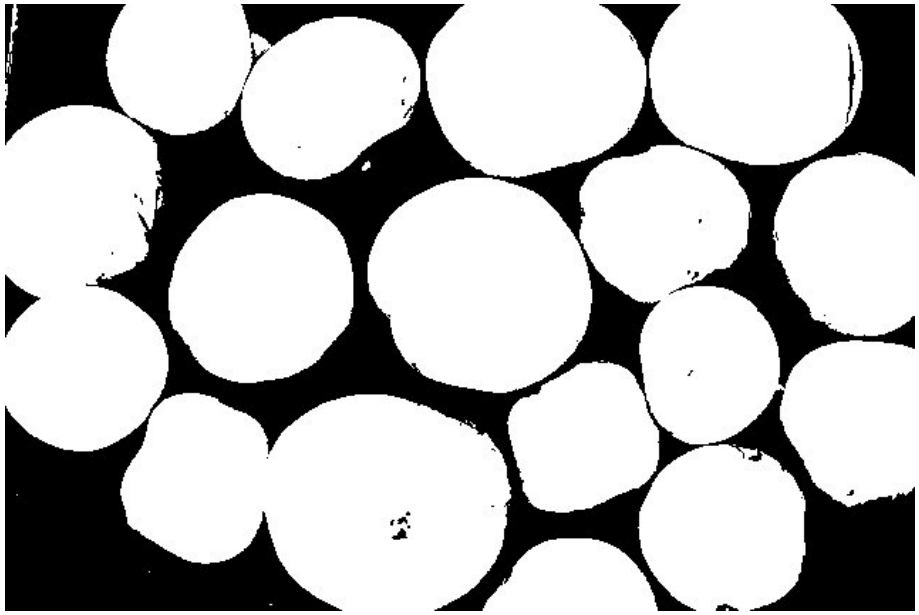


Image Segmentation

Perfect image segmentation is difficult to achieve:

- a pixel may straddle the “real” boundary of objects such that it partially belongs to two or more objects
- effects of noise, non-uniform illumination, occlusions etc. give rise to the problem of *over-segmentation* and *under-segmentation*



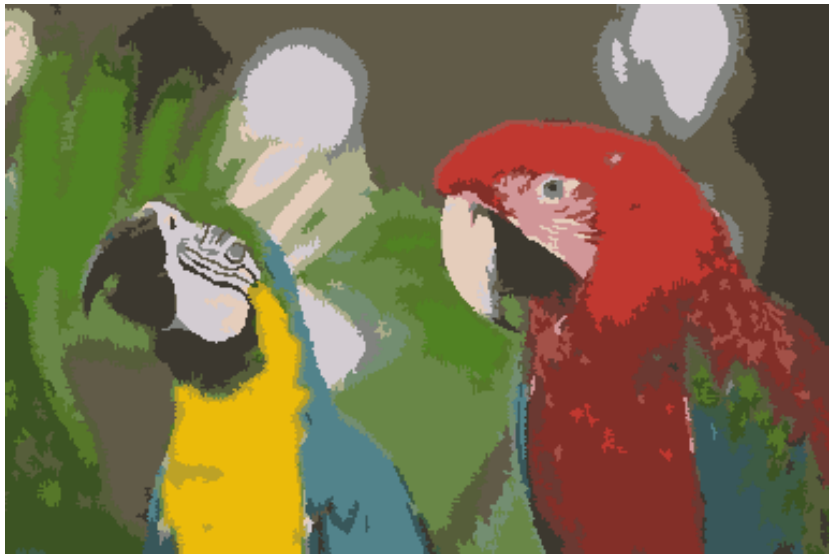


Images from craftofcoding.wordpress.com

Example of Over-Segmentation

Original image

Over-segmentation



Over-segmentation: pixels belonging to the same region [object] are classified as belonging to different regions [objects]

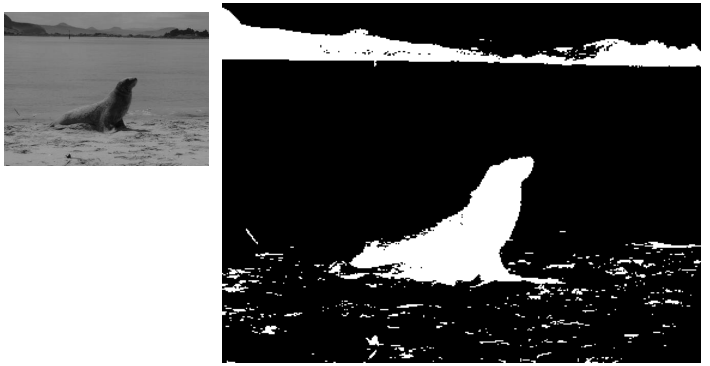
Example of Under-Segmentation

Original image

Under-segmentation

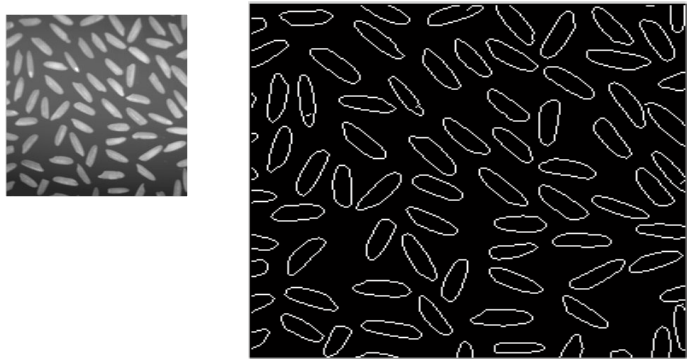
Under-segmentation: pixels belonging to different regions [objects] are classified as belonging to the same region [object]

Concepts of Segmentation I



Thresholding Methods

- pixels are categorized based on intensity
- only useful when sufficient contrast exists

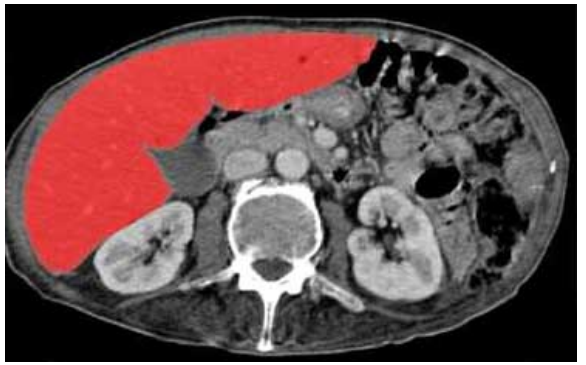


Edge-based Methods

- region boundaries are constructed from edgemaps

Region-based Methods

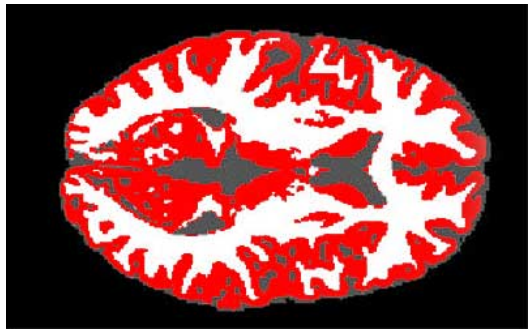
- region growing from seed pixels
- region splitting and merging for efficient spatial encoding



Concepts of Segmentation II

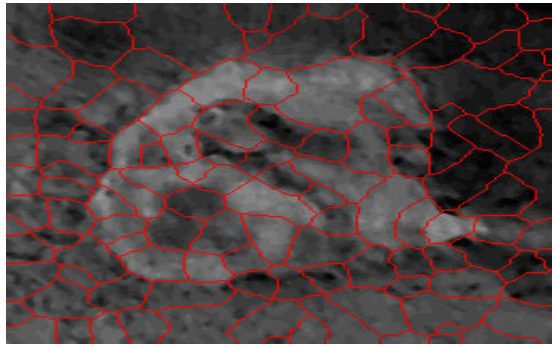
Clustering and Statistical Methods

- global, often histogram based image partitioning, e.g. *K*-means, Gaussian Mixture Model

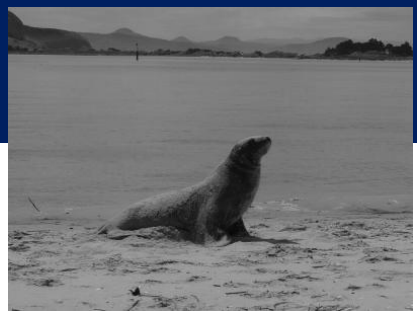


Topographic Methods **(out of scope in this unit)**

- stepwise simplifications that take spatially wider (topographical) image configurations into account e.g. watershed transform, variational based methods



Thresholding Example



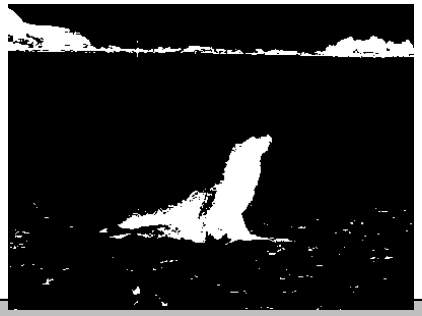
- If the image contains a dark object on a light background
  - choose a threshold value, *T*
  - for each pixel
    - if the brightness at that pixel is less than *T* it is a pixel of interest
    - otherwise it is part of the background

T = 128

T = 96

- The value of the threshold is very important
  - if too high → background pixels classified as foreground
  - If too low → foreground pixels classified as background

T = 64



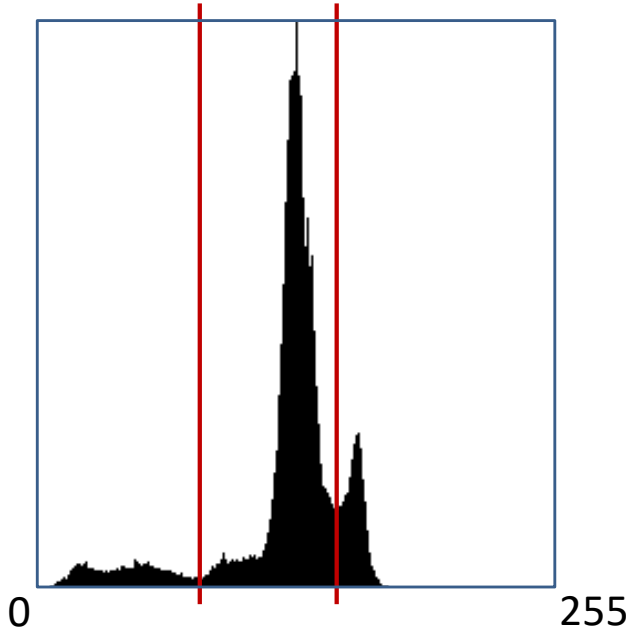
Using Histograms to Stipulate Regions

To find a threshold, we can use an image histogram:

- count how many pixels in the image have each value
- for simple images it shows peaks and valleys around regions of the image

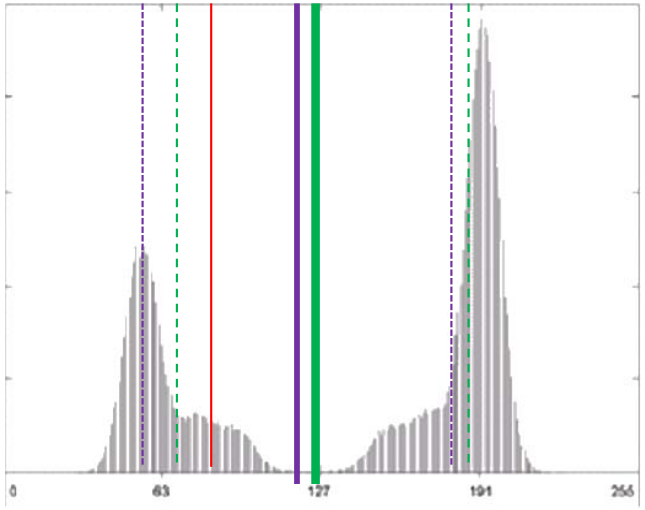
The seal image shows three regions

- one below $T_1 = 80$
- one above $T_2 = 142$
- one between the two thresholds



Threshold Selection Algorithm

1. Select an initial estimate for the threshold T
2. Segment the image using T.
   This will produce two groups of pixels: G_1 consisting of all pixels with grey levels $>T$ and G_2 consisting of pixels with grey values $<T$.
3. Compute the average grey level values m_1 and m_2 for the pixels in regions G_1 and G_2.
4. Compute a new threshold value: $T = (m_1 + m_2)/2$
5. Repeat steps *(2.)* through *(4.)* until convergence



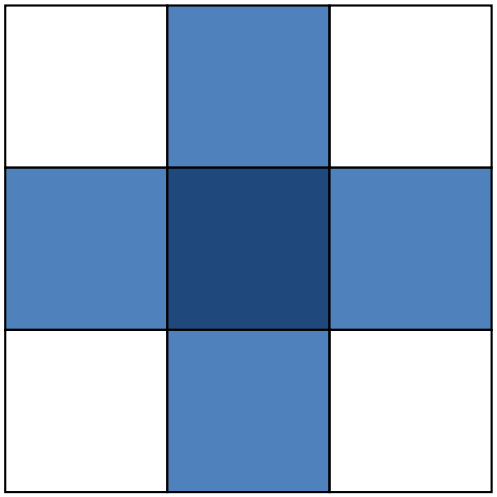
initial estimate
average values (round 1)
average values (round 2)
threshold after round 1
threshold after round 2

Connectivity

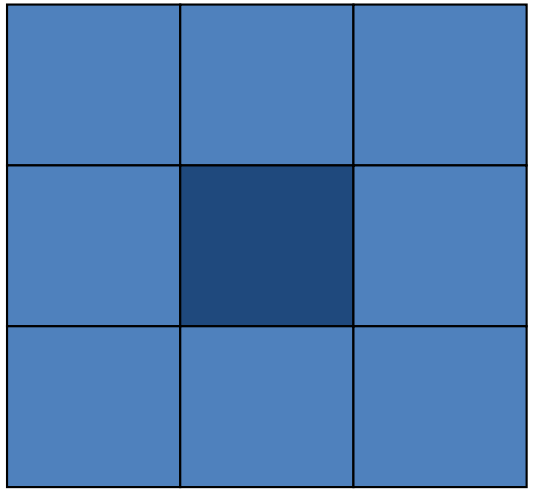
General idea:

There are several ways of defining the adjacent neighbourhood of a pixel given the image grid.

Two common paradigms for connectivity



4-connectivity



8-connectivity

Edge-based Segmentation I

General idea:

- detect strong edges
- extend or delete them in order to create closed boundaries that represent objects

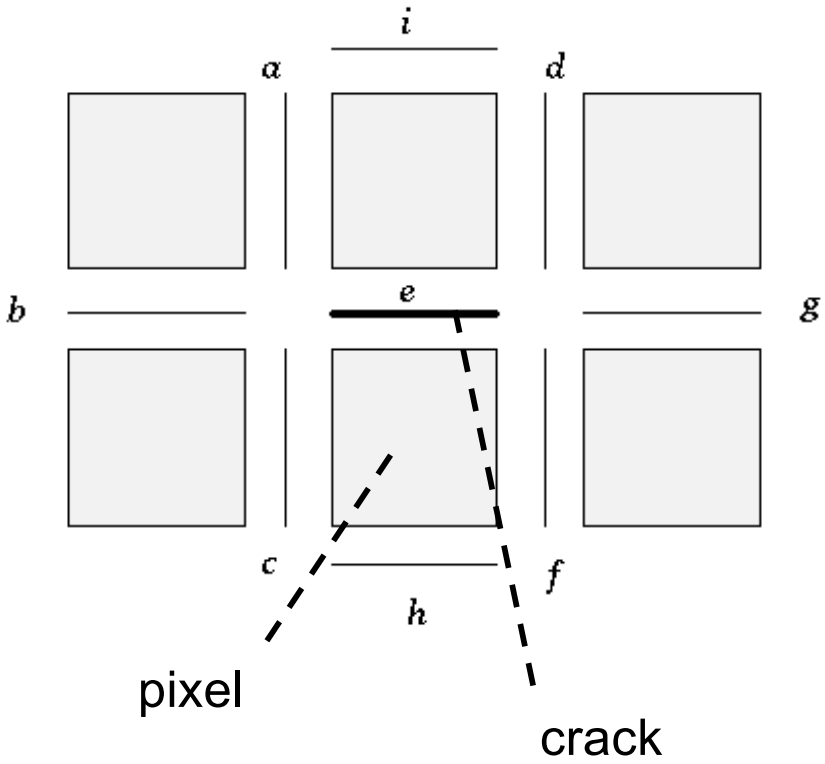
- problems: noise or no edge presence where a real object border exits

Edge-based Segmentation II

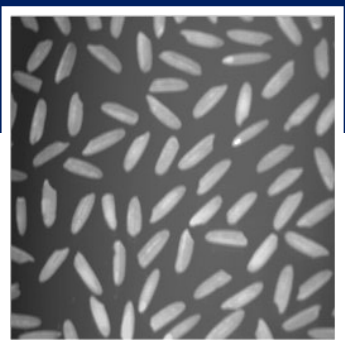
Conceptual examples:

- A weak edge positioned between two strong edges provides an example of context; it is highly probable that this inter-positioned weak edge should be a part of a resulting boundary.

- If, on the other hand, an edge (even a strong one) is positioned by itself with no supporting context, it is probably not a part of any border.

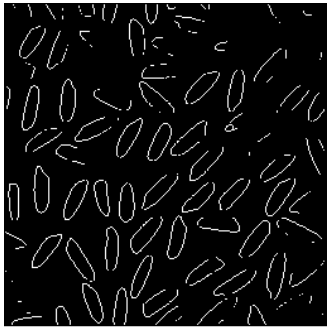


Edge-based Segmentation III

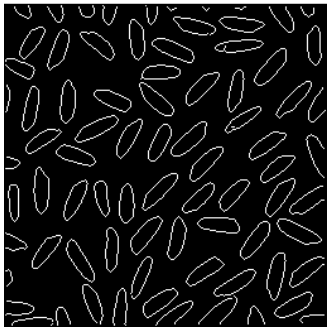


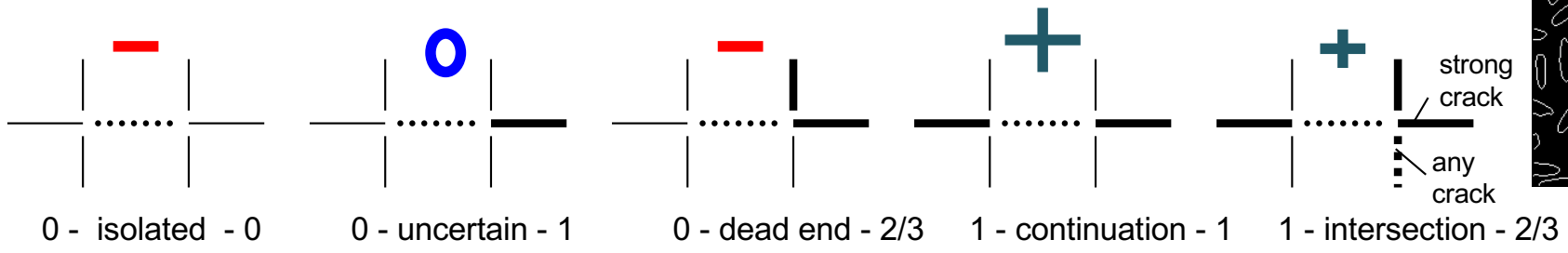
Example: Edge Relaxation

Consider each crack context (e.g. the adjacent 6 cracks) in order to iteratively establish stable, connected and closed edges, i.e. based on the strength of the edges in a specified neighbourhood, the confidence of each edge, e.g. C(e), is increased or decreased.



Relevant context-configurations and their influence on edge probability:



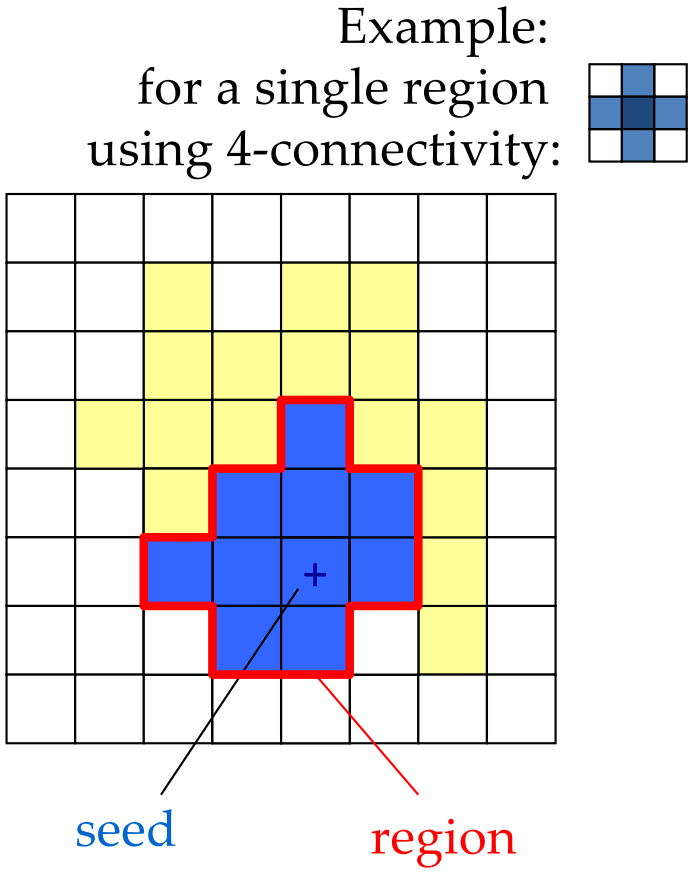


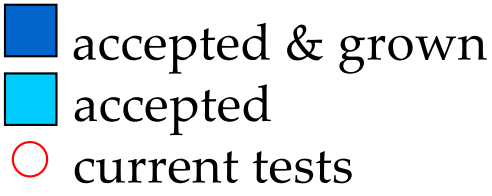
Region Growing I

Find regions by growing homgeneous areas around initially chosen seed pixels.

Region Growing Algorithm:

- Start with an initial seed pixel.
- Grow area to neighbouring pixels (based on connectivity) if they satisfy a homogeneity condition test.
- If the region doesn't grow anymore select another seed and repeat the process until all pixels are accounted for.
- Final tidying operation is often performed to remove very small regions.





Region Growing II

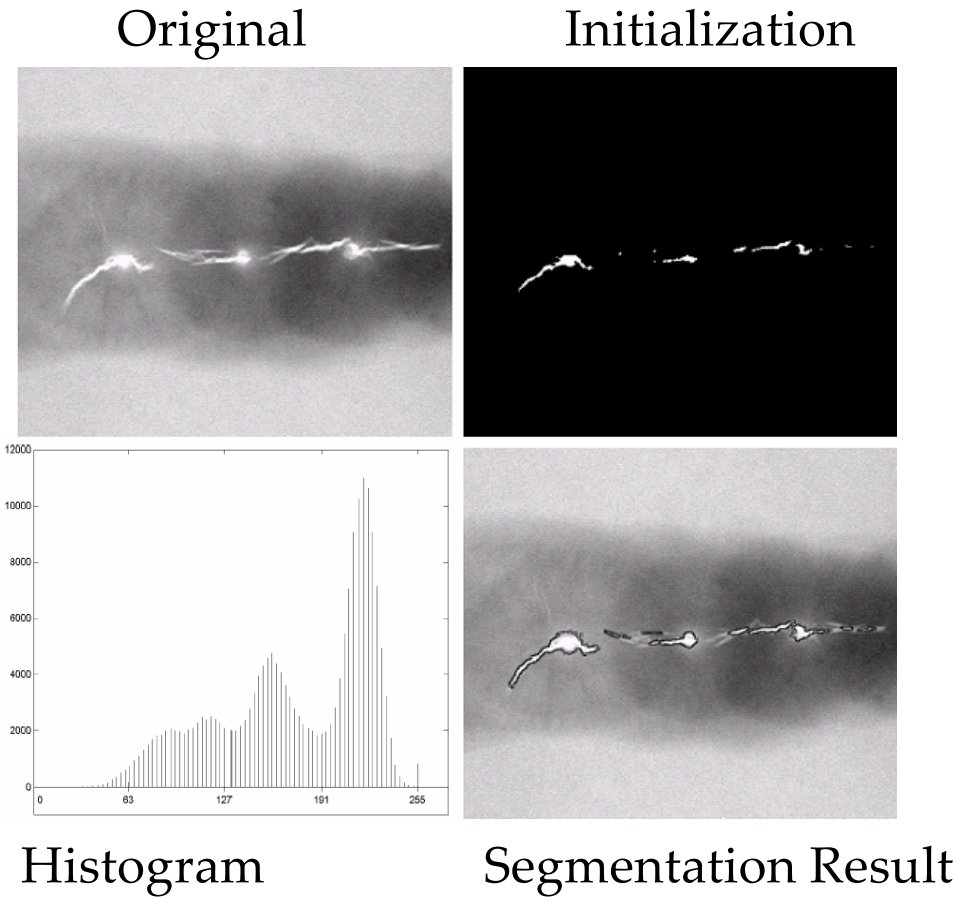
Homogenity Condition:

A characteristic function H that maps from parameters of the current region r and a new candidate pixel p to a binary decision whether to merge or not:

$$(r, p) \longrightarrow^{H} \{0, 1\}$$

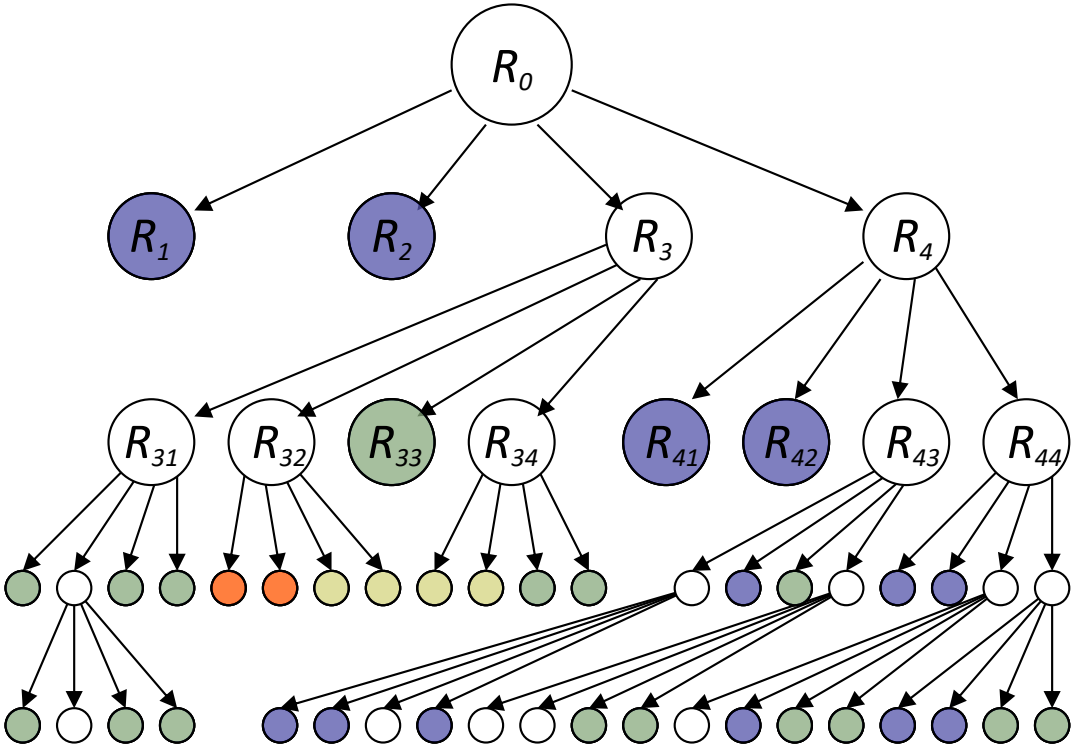
Simple Example:

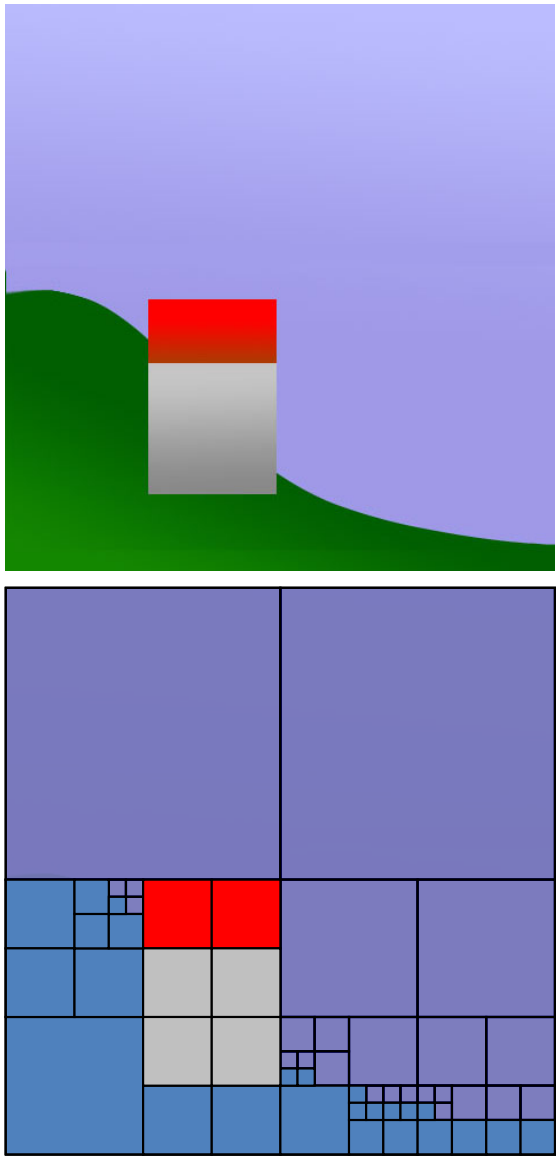
- Initialise seed region as grey level 255 (full luminance)
- Use 8-connectivity
- Basic homogeneity condition H:
  - Merge, if the luminance difference between the seed and a pixel is less than 65



Split & Merge – Divide & Conquer

1. Start with R_0 that represents the entire image
2. If $H(R_i) = 0$ (inhomogeneous) then
   {split area into 4 blocks (quadtree splitting) and
   process each area with step (2.)}
3. Merge all subregions that pairwise satisfy
   $H(R_i \cup R_j) = 1$ (homogenous)





Segmentation Result

Split & Merge – Summary

Conceptual Summary:

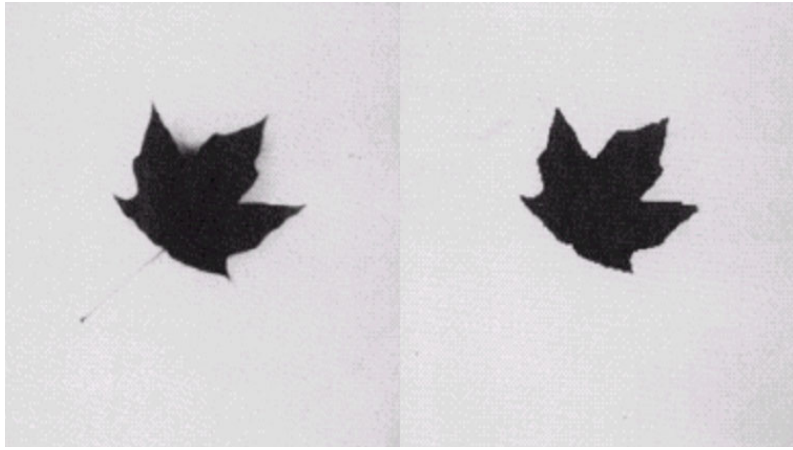
- Iteratively decompose an image into regions of a maximally sized selected shape (e.g. rectangle) that do not satisfy a homogeneity condition. (split step)
- Then merge regions that together satisfy a homogeneity condition. (merge step)

Some Comments:

- Using quadtrees, the results of split and merge tend to be *blocky*.
- Can have an adaptive homogeneity condition that, for instance, changes depending on the region size.

Example:

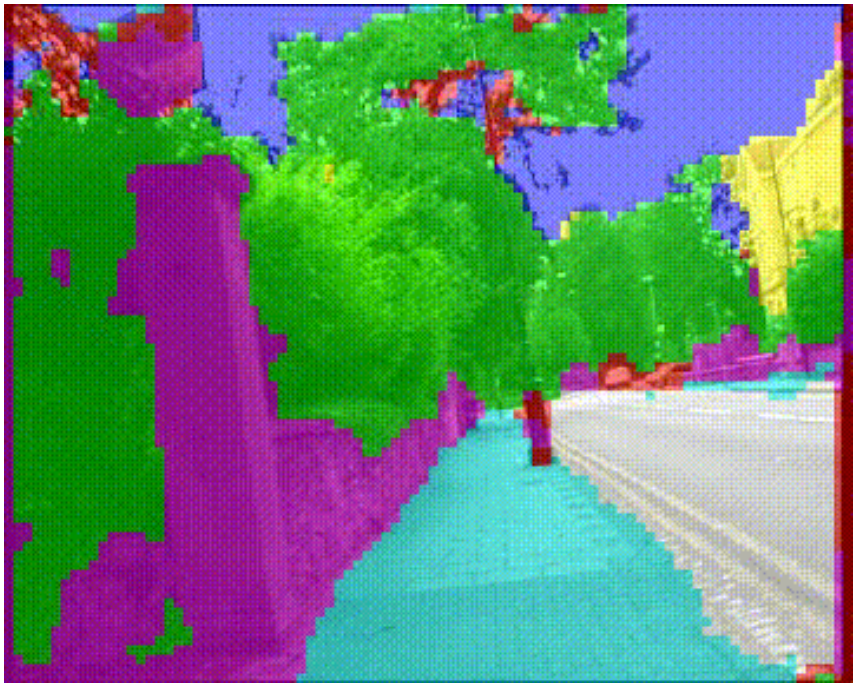
- $H(R_i)=1$ if at least 80% of the pixels in R_i have the property $|z_j - m_i| < 2\sigma_i$, where z_j is the grey level of the j^{th} pixel in R_i, m_i is the mean grey level of the region and σ_i is the standard deviation of the grey levels in R_i
- If $H(R_i)=1$ then set all the pixels in R_i to value m_i



Original Result

Split & Merge – Example

Original Video

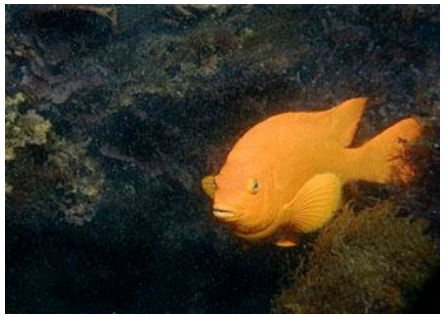


Segmentation Result

- Images are segmented using a Split-And-Merge technique. (Note the blocky nature of the regions!)

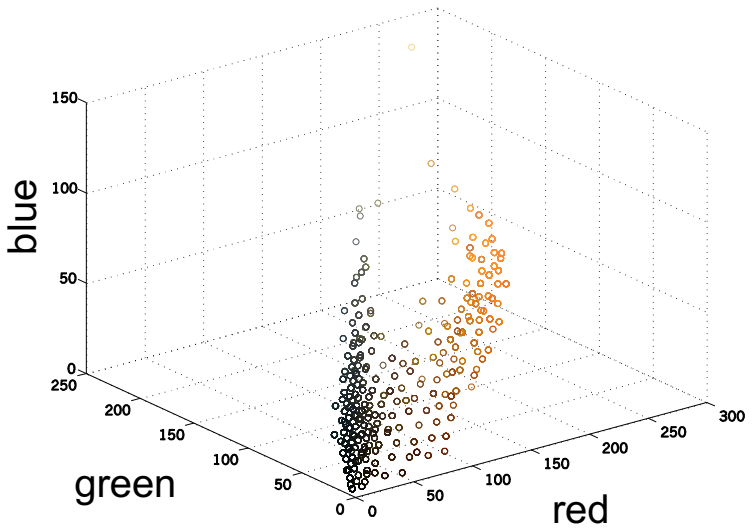
- Regions are then labelled by a Neural Network to associate the segments with semantics (colouration).

Clustering for image segmentation

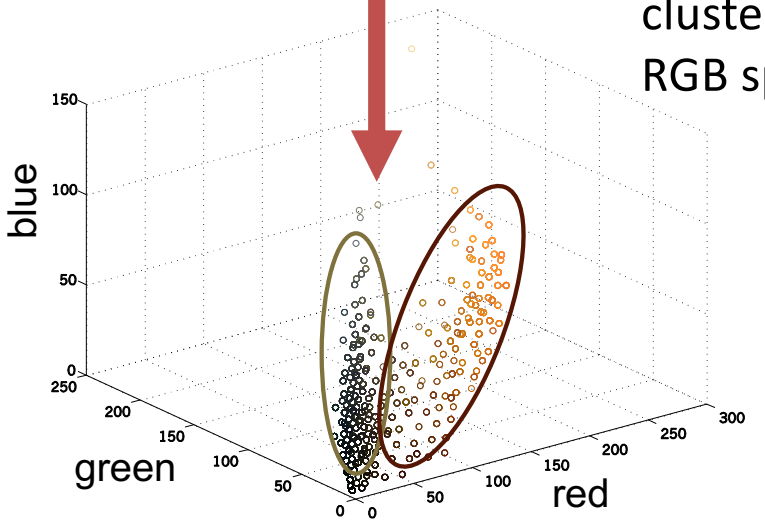


map to 3D

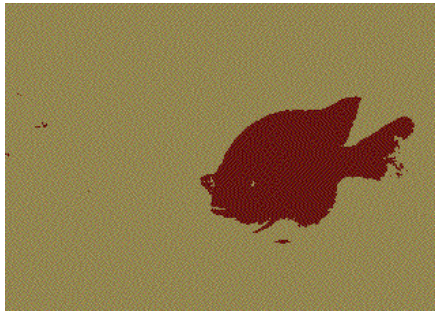
RGB space



clustering in RGB space



map back to pixel space



K-means clustering – theoretical view

- We are effectively minimising the following objective function:

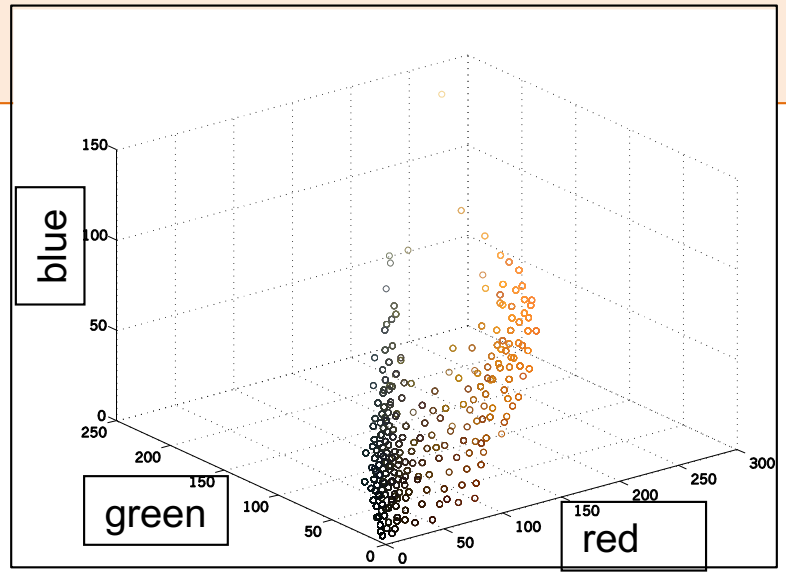
$$\Theta(clusters, data) = \sum_{j \in clusters} \left[\sum_{i \in j^{th} cluster} \left\| \mathbf{x}_i - \boldsymbol{\mu}_j \right\|^2 \right]$$

- ***K*-means iterates through two activities…**
  - Given current cluster centres, allocates each point to the closest cluster centre
  - Given the current allocation, chooses a new set of cluster centres, with each centre being the mean of the points allocated to that cluster.
- **....until it converges to a local minimum of the objective function**.

- The starting *K* points are chosen randomly.

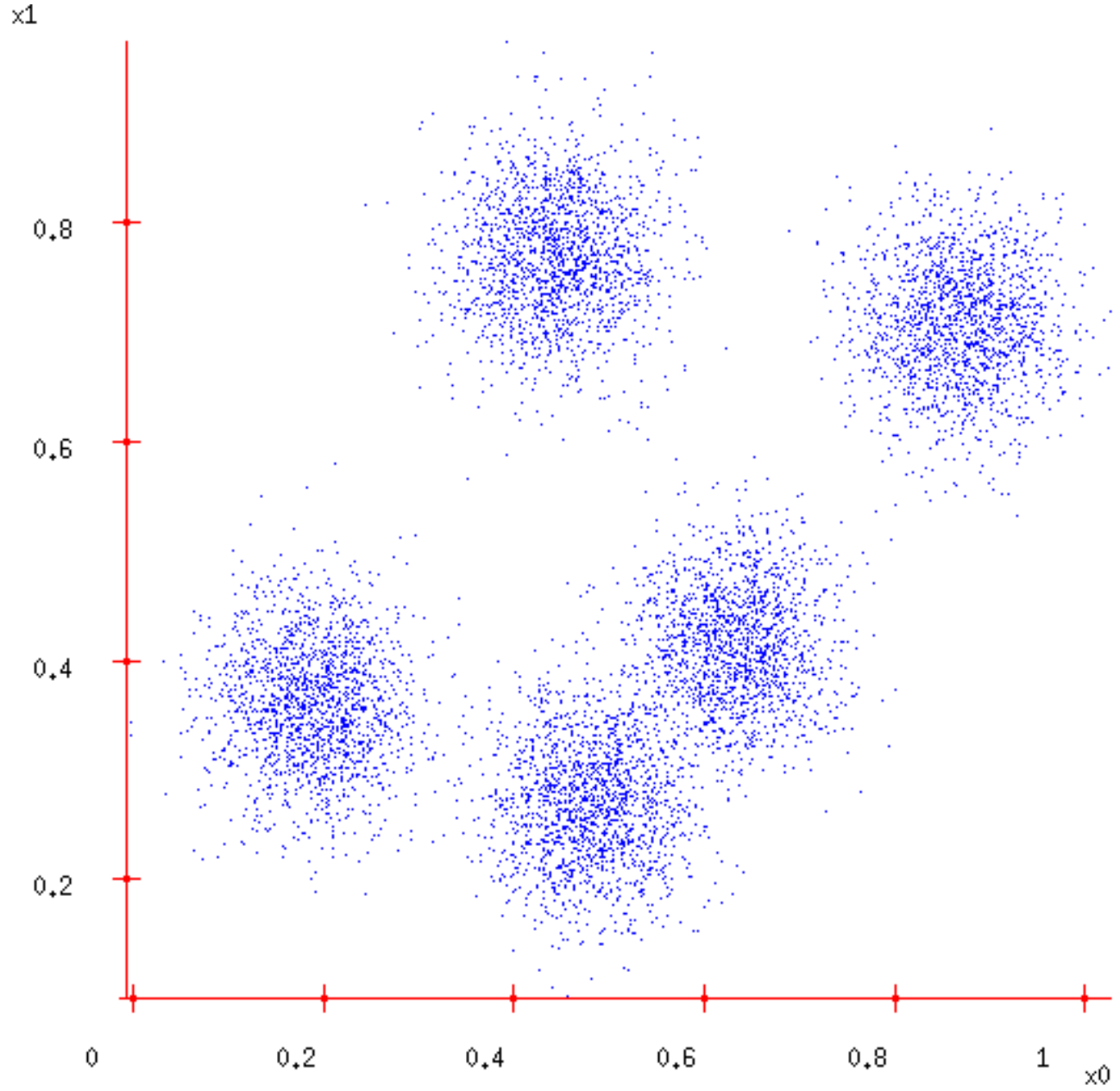
K-means clustering

function KMeans(*Features*,*K*)
  randomly initialise *K* vectors $\mu_1 \ldots \mu_K$;
  repeat
    assign each $x \in$ *Features* to the nearest μ_j;
    recompute each μ_j as the mean of the features assigned to it;
  until no change in $\mu_1 \ldots \mu_K$;
  return $\mu_1 \ldots \mu_K$;



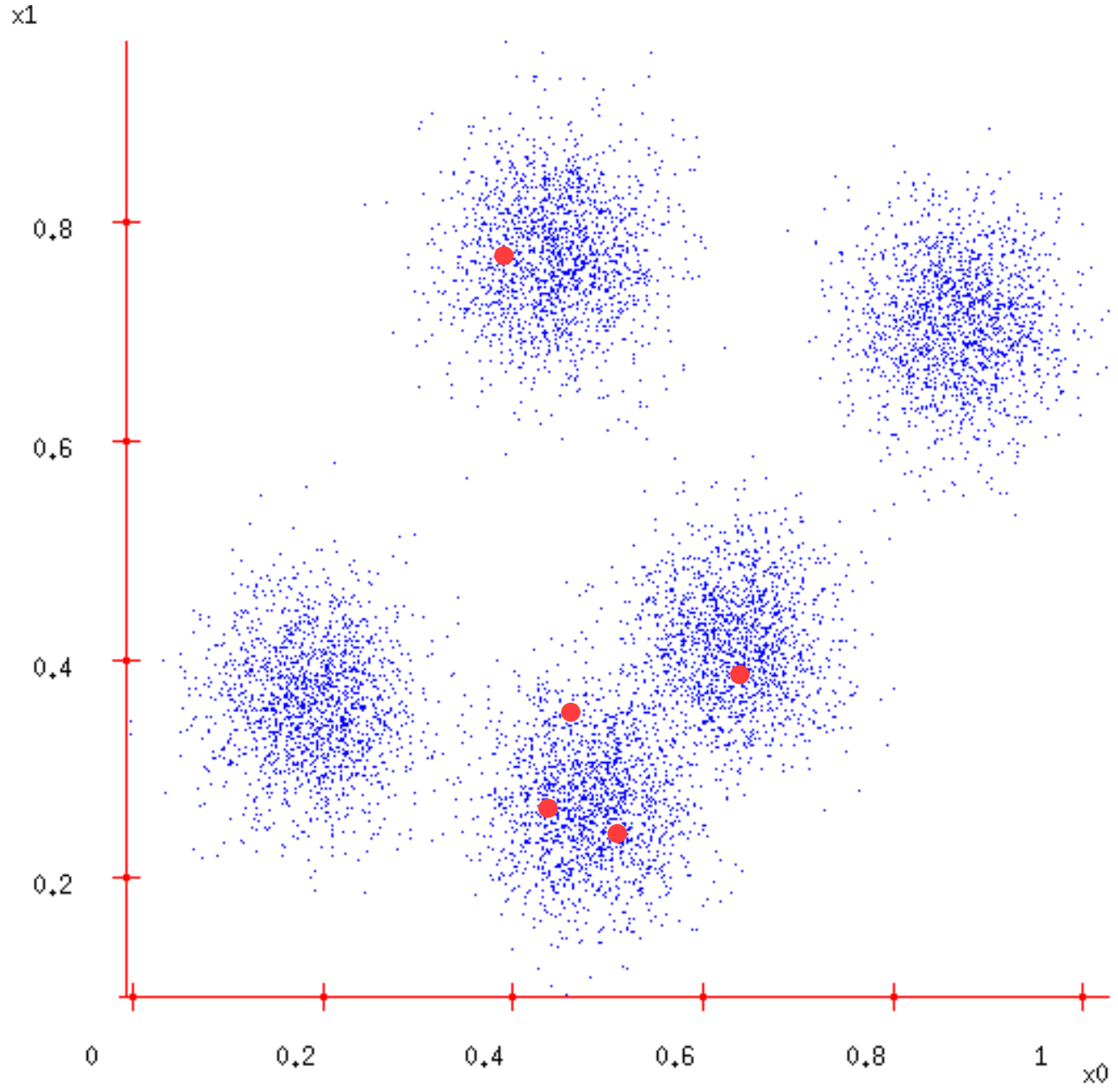
K-means clustering

1. Ask user how many clusters they'd like (e.g., *K*=5)



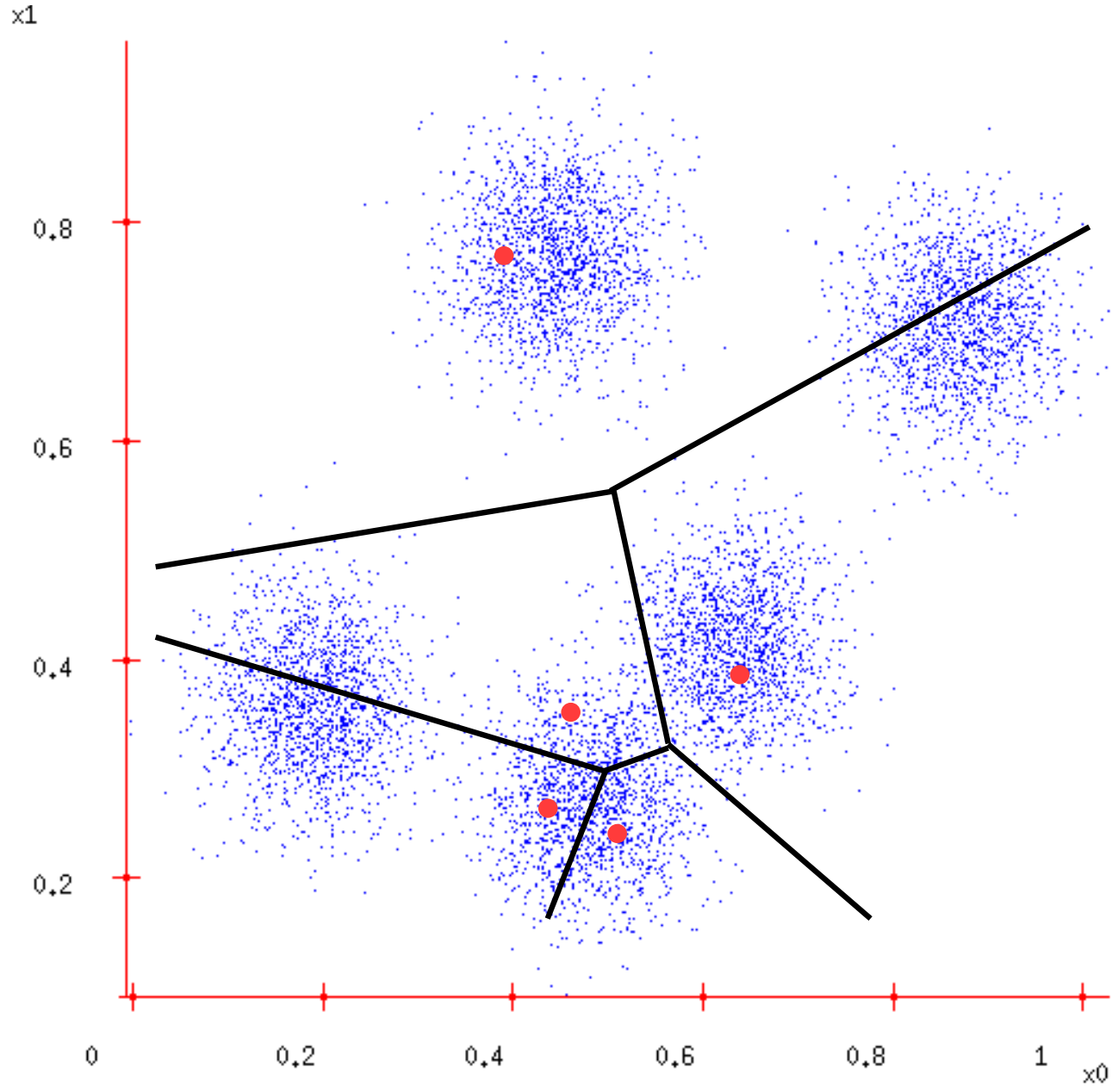
K-means clustering

1. Ask user how many clusters they'd like (e.g., K=5)
2. Randomly guess K cluster centre locations (μ_1 … μ_K)



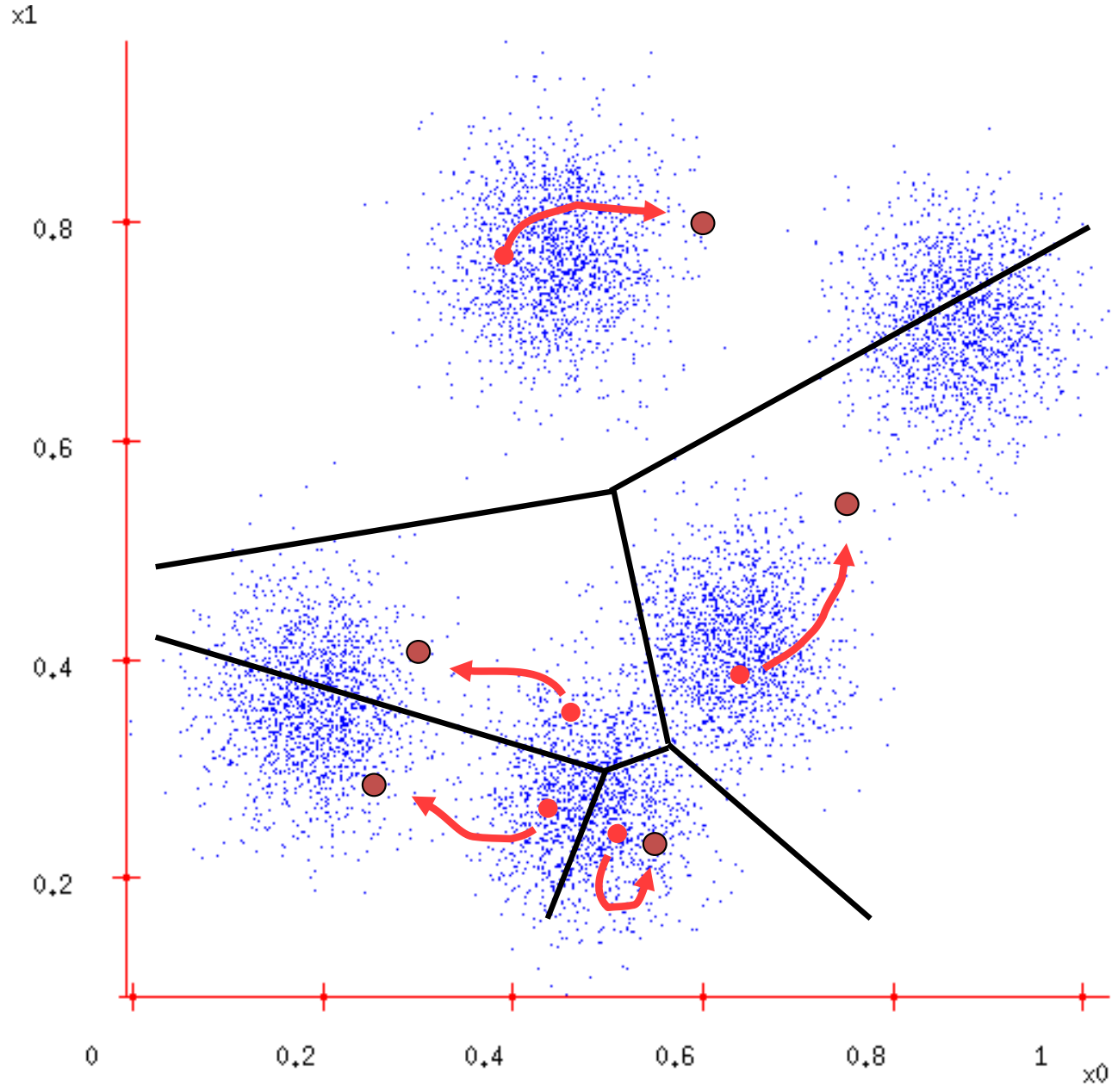
K-means clustering

1. Ask user how many clusters they'd like (e.g., K=5)
2. Randomly guess K cluster centre locations (μ_1 … μ_K)
3. Each datapoint finds out which centre it's closest to (thus each centre "owns" a set of datapoints)



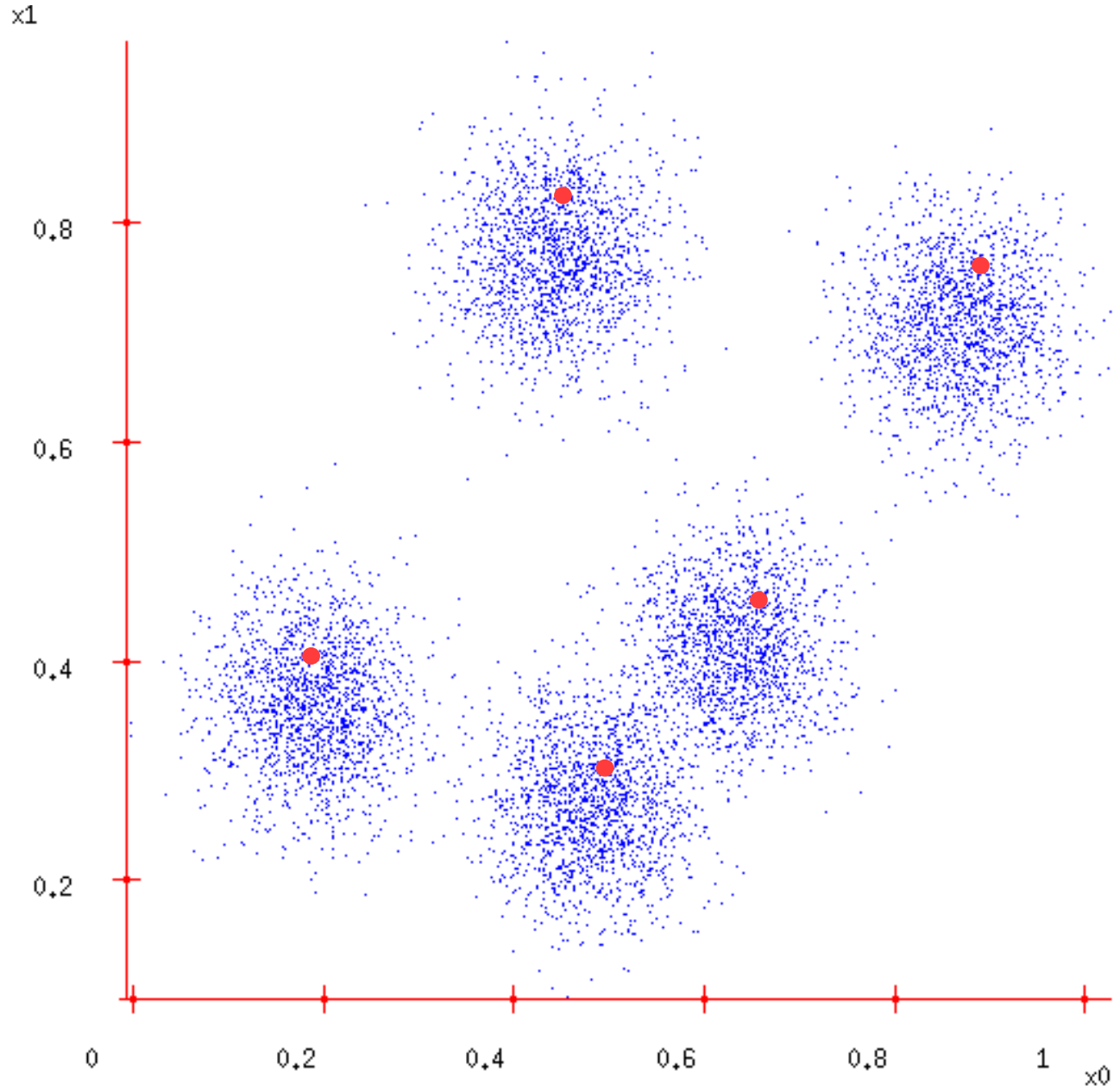
K-means clustering

1. Ask user how many clusters they'd like (e.g., K=5)
2. Randomly guess K cluster centre locations (μ_1 … μ_K)
3. Each datapoint finds out which centre it's closest to
4. Each centre finds the centroid of the points it owns…
5. …and jumps there



K-means clustering

1. Ask user how many clusters they'd like (e.g., K=5)
2. Randomly guess K cluster centre locations (μ_1 … μ_K)
3. Each datapoint finds out which centre it's closest to
4. Each centre finds the centroid of the points it owns…
5. …and jumps there
6. Repeat from 3 until terminated!



Reflection on the *K*-means Algorithm

- **What does it do?**
  - *K*-means attempts to find a configuration $\mu_1 \ldots \mu_K$ that minimises within-cluster scatter: total squared distance between point x_i and centroid μ_j in j^{th} cluster:

$$\sum_i \left\| \mathbf{x}_i - \boldsymbol{\mu}_j \right\|^2$$

  - This is equivalent to maximising the between-cluster scatter (total squared distance between each cluster centroid and the global centroid of all points)
- **Does it work?**
  1. The algorithm terminates.
  2. It finds a local optimum from which no further improvement is possible by making local changes.
  3. It does not necessarily find a global optimum.