Clustering

Artificial Intelligence
School of Computer Science
The University of Adelaide

Introduction

Clustering is the process of identifying groups, or clusters, of data points in a (usually) multidimensional space.

It is a type of **unsupervised learning** method.

Formally, suppose we have N points $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, where each $\mathbf{x}_i$ is a multidimensional vector of length M.

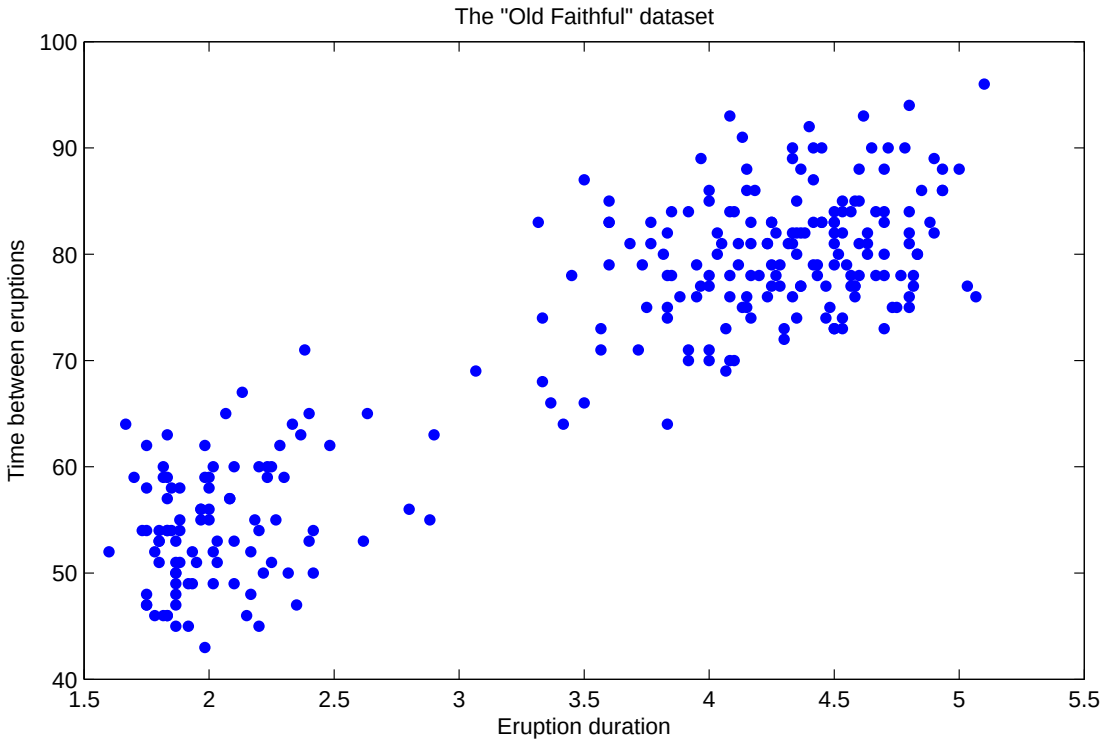
Our goal is to partition X into K clusters.

Intuitively, we may think of a cluster as comprising a group of data points whose inter-point distances are small compared with the distances to points outside of the cluster.

Introduction (cont.)

Example:

The "Old Faithful" dataset which records the eruption duration and time between eruptions of the Old Faithful geyser in Yellowstone NP.



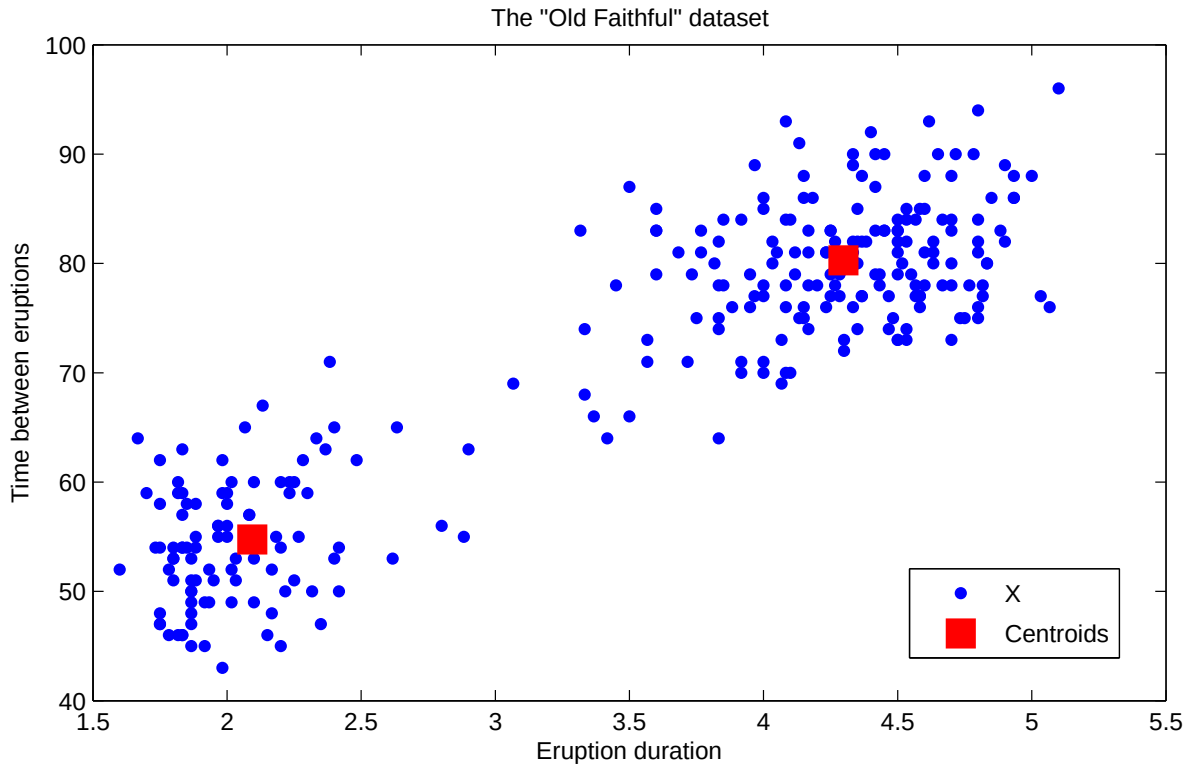
How many clusters do you see? Which points belong to which cluster?

The K-means Algorithm

Centroids

A cluster is represented by a prototype point called its **centroid**.

Let the K centroids be $\boldsymbol{\mu}_k$, $k = 1, 2 \ldots, K$, where $\boldsymbol{\mu}_k$ is the centroid of the k-th cluster. Each $\boldsymbol{\mu}_k$ is also a vector of M dimensions.



Point assignments

Given a set of K centroids, we can introduce a corresponding set of **binary indicator** variables $r_{i,k}$ defined as follows:

$$r_{i,k} = \begin{cases} 1 & \text{if } \mathbf{x}_i \text{ belongs to cluster } k \\ 0 & \text{otherwise} \end{cases}$$

Hence, there are a total of $K \times N$ of these indicator variables.

Note that the binary nature of these variables means that for each i, variable $r_{i,k}$ can be 1 only for one of the k's.

This is another example of the **1-of-K coding scheme**.

In other words, a point $\mathbf{x}_i$ can be assigned to only one of the K clusters.

Point assignments (cont.)

To compute the point assignments, we will have to compute the distance between a point and a potential cluster centre.

We use the Euclidean distance:

$$d(\mathbf{z}, \mathbf{x}) = \sqrt{\sum_{j=1}^{M} (\mathbf{z}[j] - \mathbf{x}[j])^2}$$

Thus, given K centroids, the $r_{i,k}$ for a point $\mathbf{x}_i$ is computed as

$$r_{i,k} = \begin{cases} 1 & \text{if } d(\mathbf{x}_i, \boldsymbol{\mu}_k) = \min\{d(\mathbf{x}_i, \boldsymbol{\mu}_1), d(\mathbf{x}_i, \boldsymbol{\mu}_2), \ldots, d(\mathbf{x}_i, \boldsymbol{\mu}_K)\} \\ 0 & \text{otherwise} \end{cases}$$

i.e. point $\mathbf{x}_i$ is assigned to the centroid with which it has the smallest distance.

Example

Let

$$X = \left\{ \begin{bmatrix} 6.2 \\ 7.3 \end{bmatrix}, \begin{bmatrix} 2.6 \\ 2.6 \end{bmatrix}, \begin{bmatrix} 6.7 \\ 6.5 \end{bmatrix}, \begin{bmatrix} 5.8 \\ 6.4 \end{bmatrix}, \begin{bmatrix} 6.2 \\ 5.2 \end{bmatrix}, \begin{bmatrix} 3.4 \\ 3.3 \end{bmatrix} \right\}$$

Let the current set of 2 centroids be

$$\boldsymbol{\mu}_1 = \begin{bmatrix} 3 \\ 5.5 \end{bmatrix}, \quad \boldsymbol{\mu}_2 = \begin{bmatrix} 6 \\ 6 \end{bmatrix}$$

We can display the distances between points and centroids as

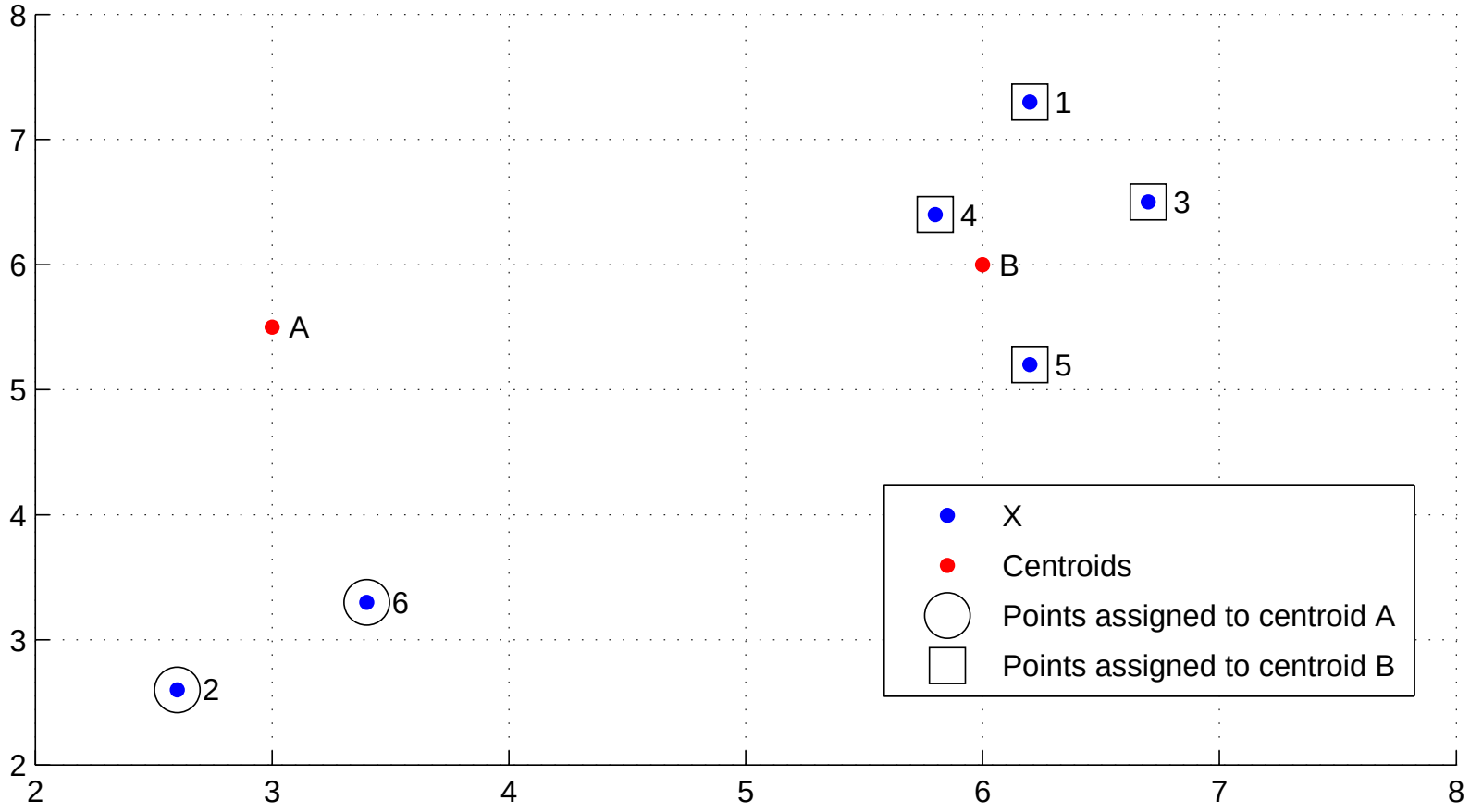
	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$	$\mathbf{x}_6$
$\boldsymbol{\mu}_1$	3.67	2.93	3.83	2.94	3.21	2.24
$\boldsymbol{\mu}_2$	1.32	4.81	0.86	0.45	0.82	3.75

This results in the following indicator variables/point assignments:

	$r_{1,k}$	$r_{2,k}$	$r_{3,k}$	$r_{4,k}$	$r_{5,k}$	$r_{6,k}$
$k = 1$	0	1	0	0	0	1
$k = 2$	1	0	1	1	1	0

Example (cont.)

This can be plotted as follows:



A chicken-and-egg problem

Clustering is a **chicken-and-egg** problem:

1. We wish to find the assignment of data points X to the K clusters. If we have the centroids this can be accomplished easily as shown previously.
2. We wish to compute the positions of the centroids. If we have the point assignments, this can be done easily by computing the k-th centroid as the mean vector of points assigned to the k-th cluster.

However, given data X we have **neither the centroids nor the point assignments**...

K-means algorithm

K-means begin by **randomly initialising the centroids**. It then alternates between two steps: (1) Assignment and (2) Update.

Require: Input data $X = \{\mathbf{x}_i\}_{i=1}^N$, number of clusters K.

1: Randomly initialise K centroids $\{\boldsymbol{\mu}_k\}_{k=1}^K$.
2: **while** Less than maximum iterations or centroids have not converged **do**
3: # Do point assignments.
4: **for** $i = 1, \ldots, N$ **do**
5: Compute distance $d(\mathbf{x}_i, \boldsymbol{\mu}_k)$, $k = 1, \ldots, K$.
6: Set k^* as the minimiser of $d(\mathbf{x}_i, \boldsymbol{\mu}_k)$, $k = 1, \ldots, K$.
7: Set $r_{i,k^*} = 1$ and $r_{i,k} = 0$ for all $k \neq k^*$.
8: **end for**
9: # Update the centroids.
10: **for** $k = 1, \ldots, K$ **do**
11: Update $\boldsymbol{\mu}_k = \frac{1}{\sum_{i=1}^N r_{i,k}} \sum_{i=1}^N r_{i,k} \mathbf{x}_i$
12: **end for**
13: **end while**

K-means algorithm (cont.)

Steps 3–7: Assignment step:

For each point, calculate the distance to all the centroids. Find the centroid the point is closest to. Set the indicator variable corresponding to the closest point to 1, and the rest as 0.

Steps 8–10: Update step:

For each centroid, compute its new position as the average vector of the points that are assigned to it.

After a sufficient number of steps, the centroids will converge to the true centroids of the data. We can stop the algorithm when the overall differences between the current centroids and previous centroids become insignificant.

It is important to note that the results of K-means is **not deterministic**. Different intialisations will give different results.

Example

From the previous example, the indicator variables are

	$r_{1,k}$	$r_{2,k}$	$r_{3,k}$	$r_{4,k}$	$r_{5,k}$	$r_{6,k}$
$k=1$	0	1	0	0	0	1
$k=2$	1	0	1	1	1	0

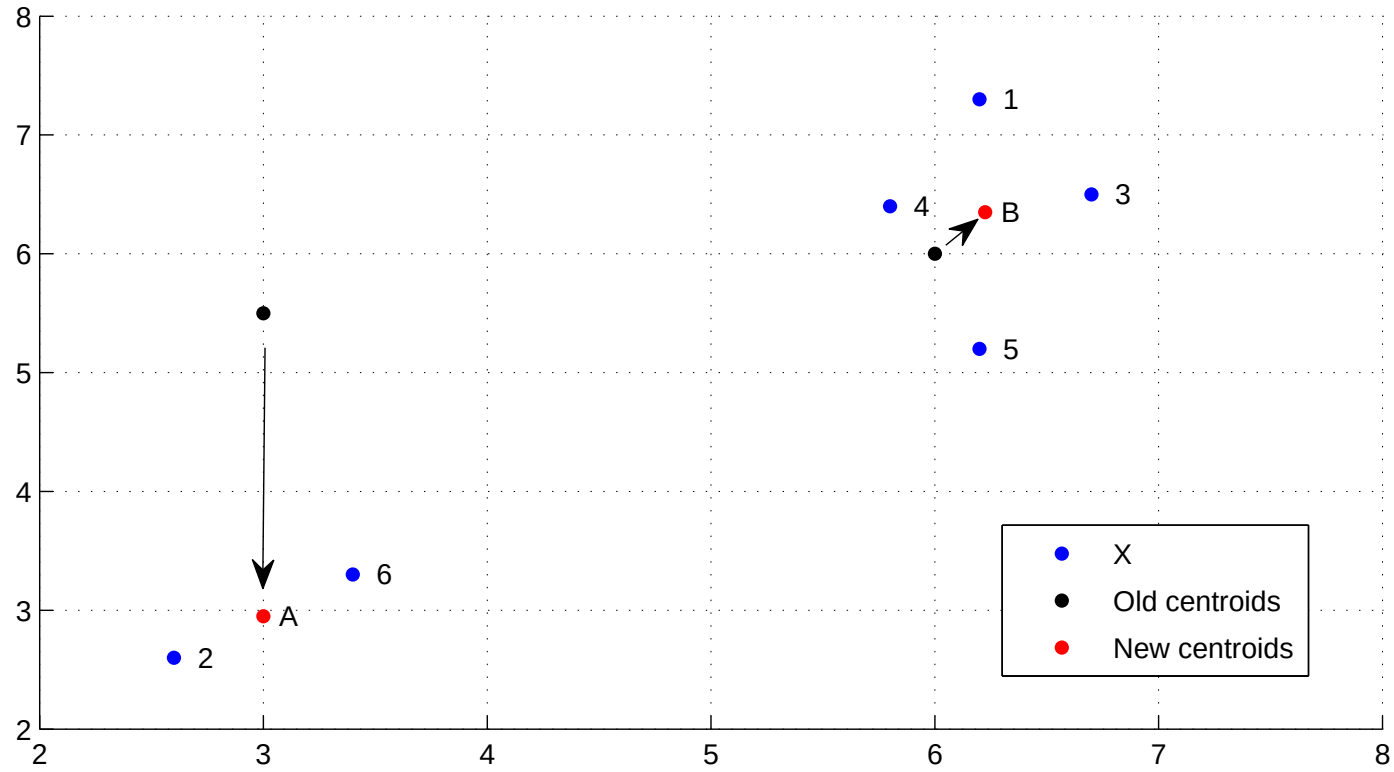
We update the centroids as

$$\boldsymbol{\mu}_1 = \frac{1}{\sum_{i=1}^{6} r_{i,1}} \sum_{i=1}^{6} r_{i,1}\mathbf{x}_i = \frac{1}{2}(\mathbf{x}_2 + \mathbf{x}_6) = \begin{bmatrix} 3 \\ 2.95 \end{bmatrix}$$

$$\boldsymbol{\mu}_2 = \frac{1}{\sum_{i=1}^{6} r_{i,2}} \sum_{i=1}^{6} r_{i,2}\mathbf{x}_i = \frac{1}{4}(\mathbf{x}_1 + \mathbf{x}_3 + \mathbf{x}_4 + \mathbf{x}_5) = \begin{bmatrix} 6.225 \\ 6.35 \end{bmatrix}$$

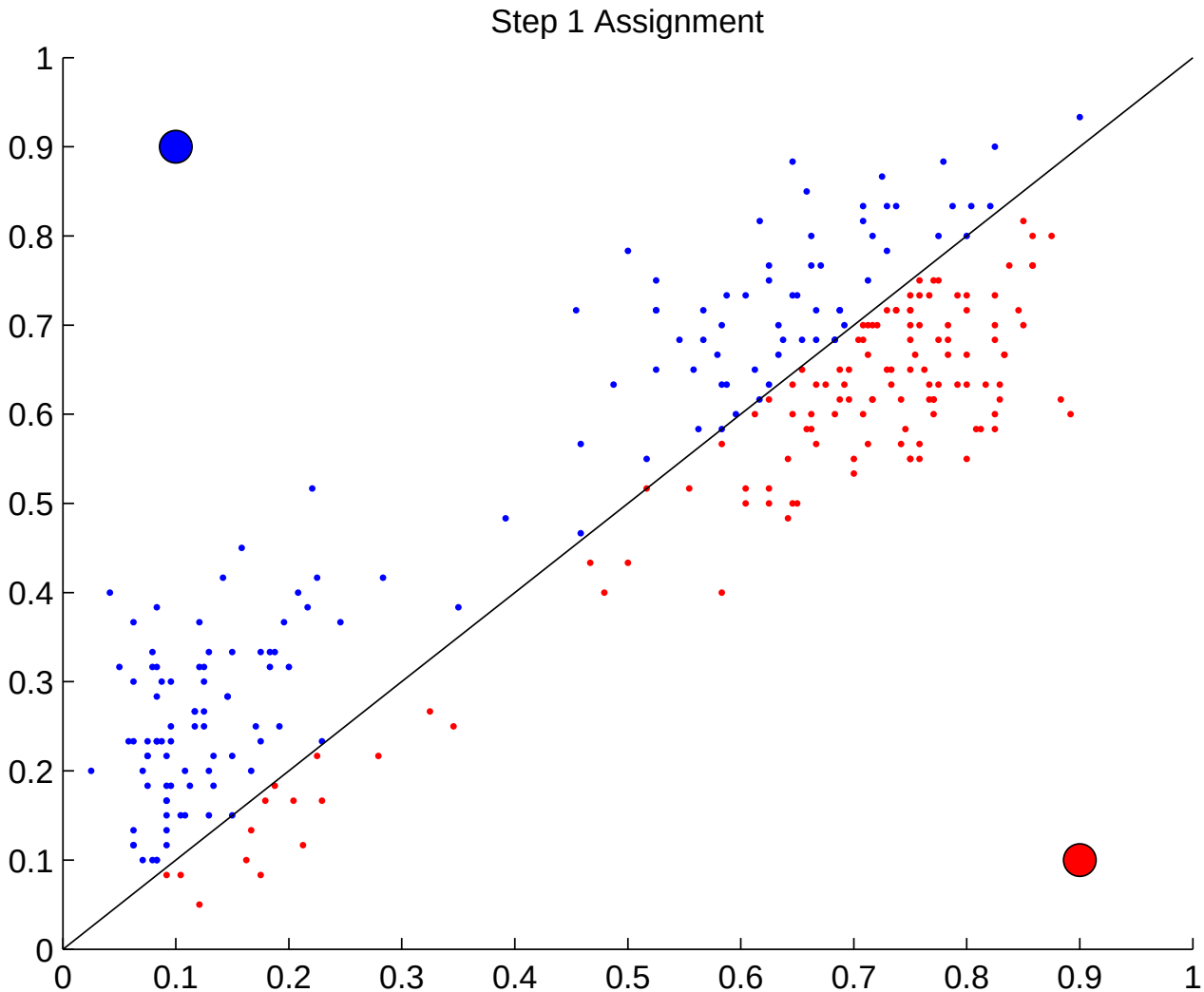
Example (cont.)

Observe that the new centroids have moved closer to the true centroids of the 2 clusters:

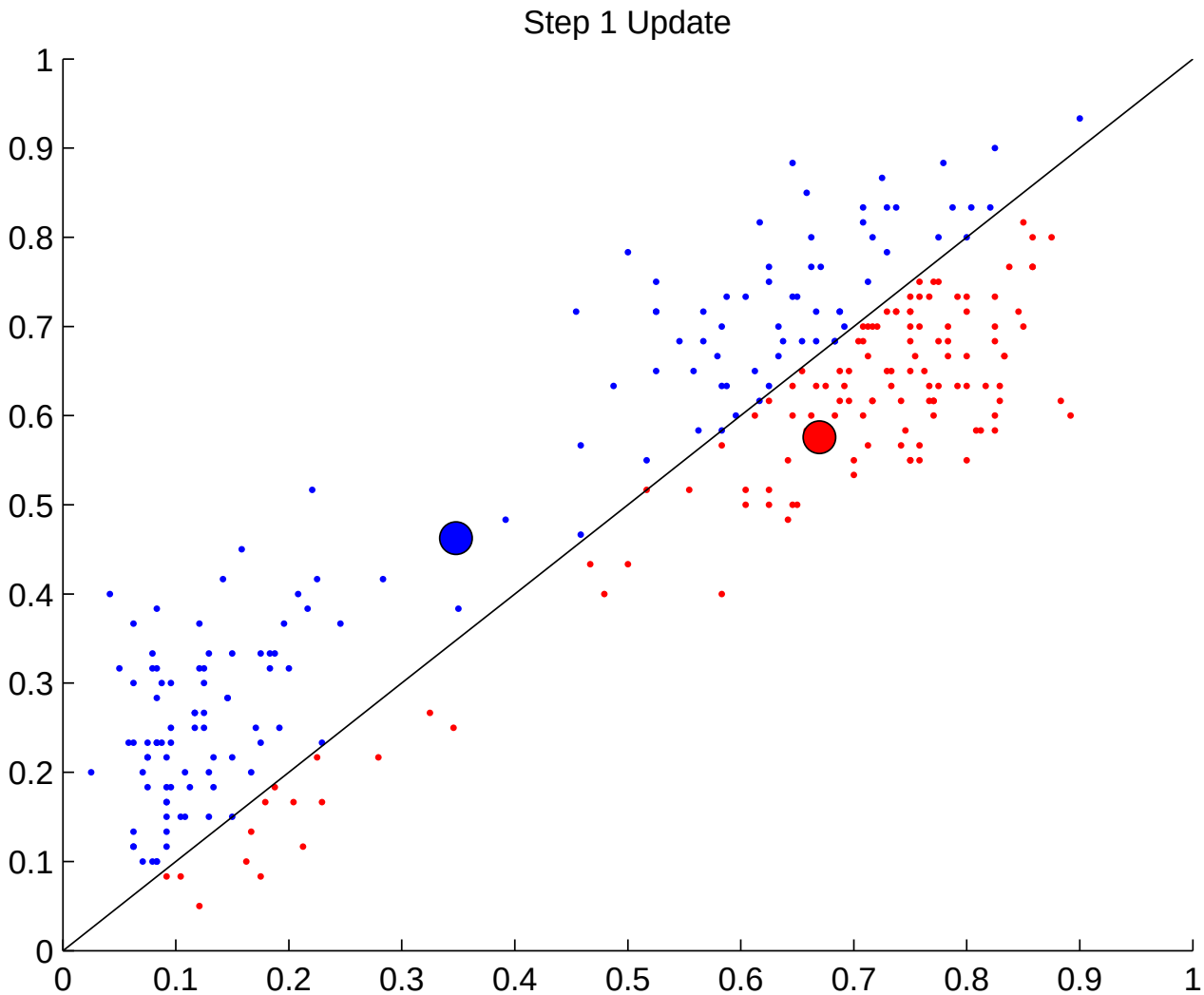


Clustering Old Faithful

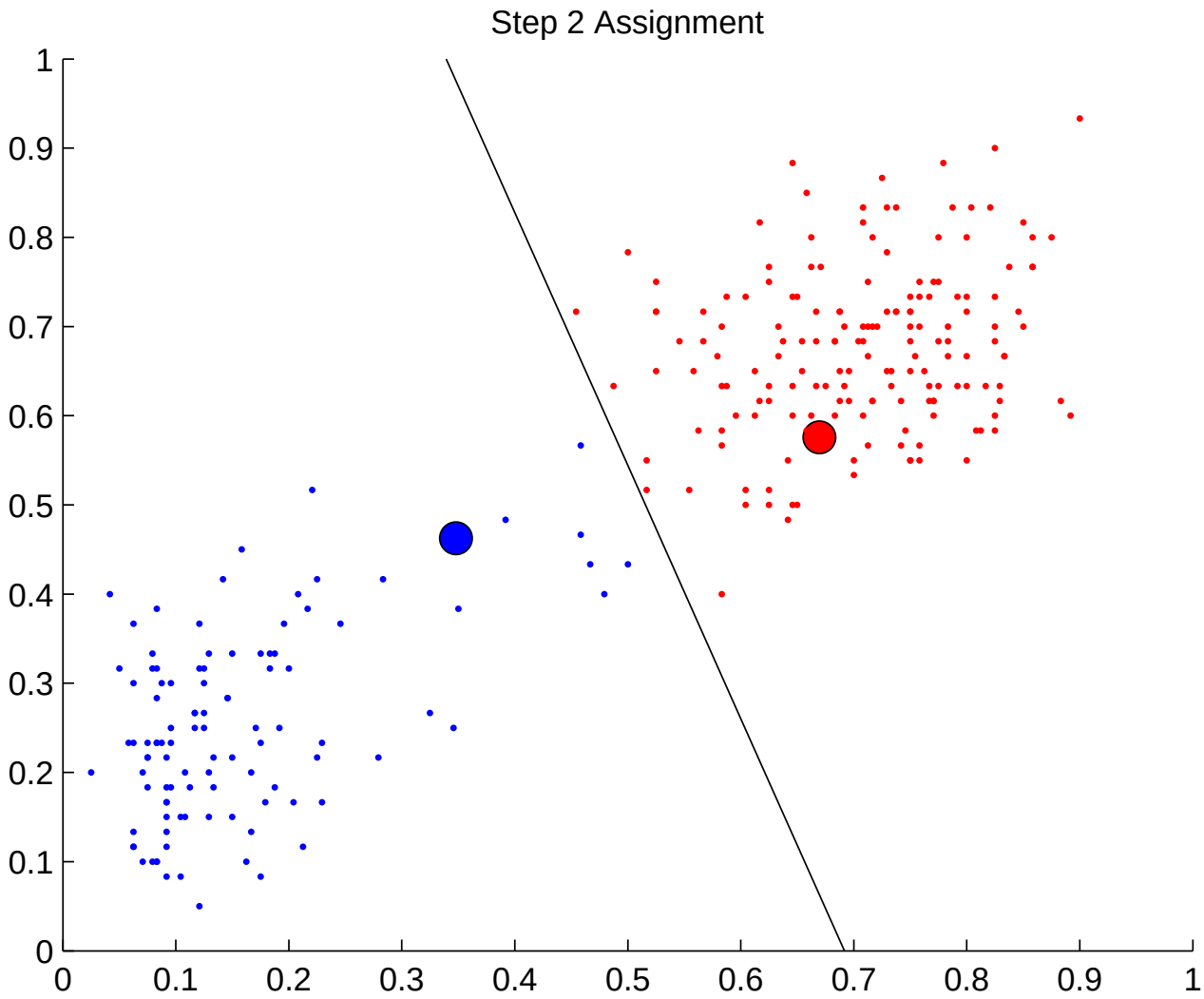
Running K-means on the Old Faithful dataset. We first normalise the data such that they are within the range of [0 1].



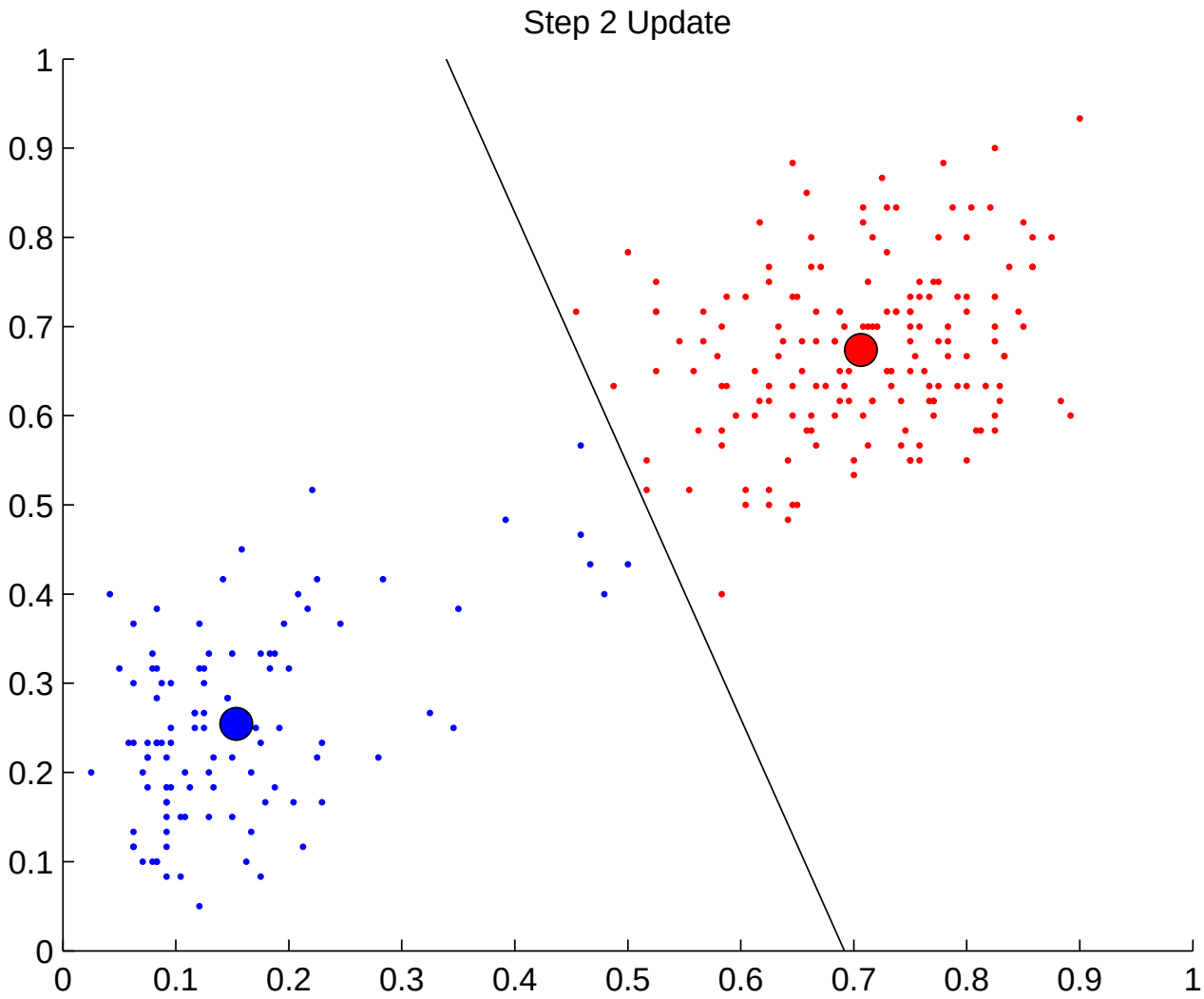
Clustering Old Faithful (cont.)



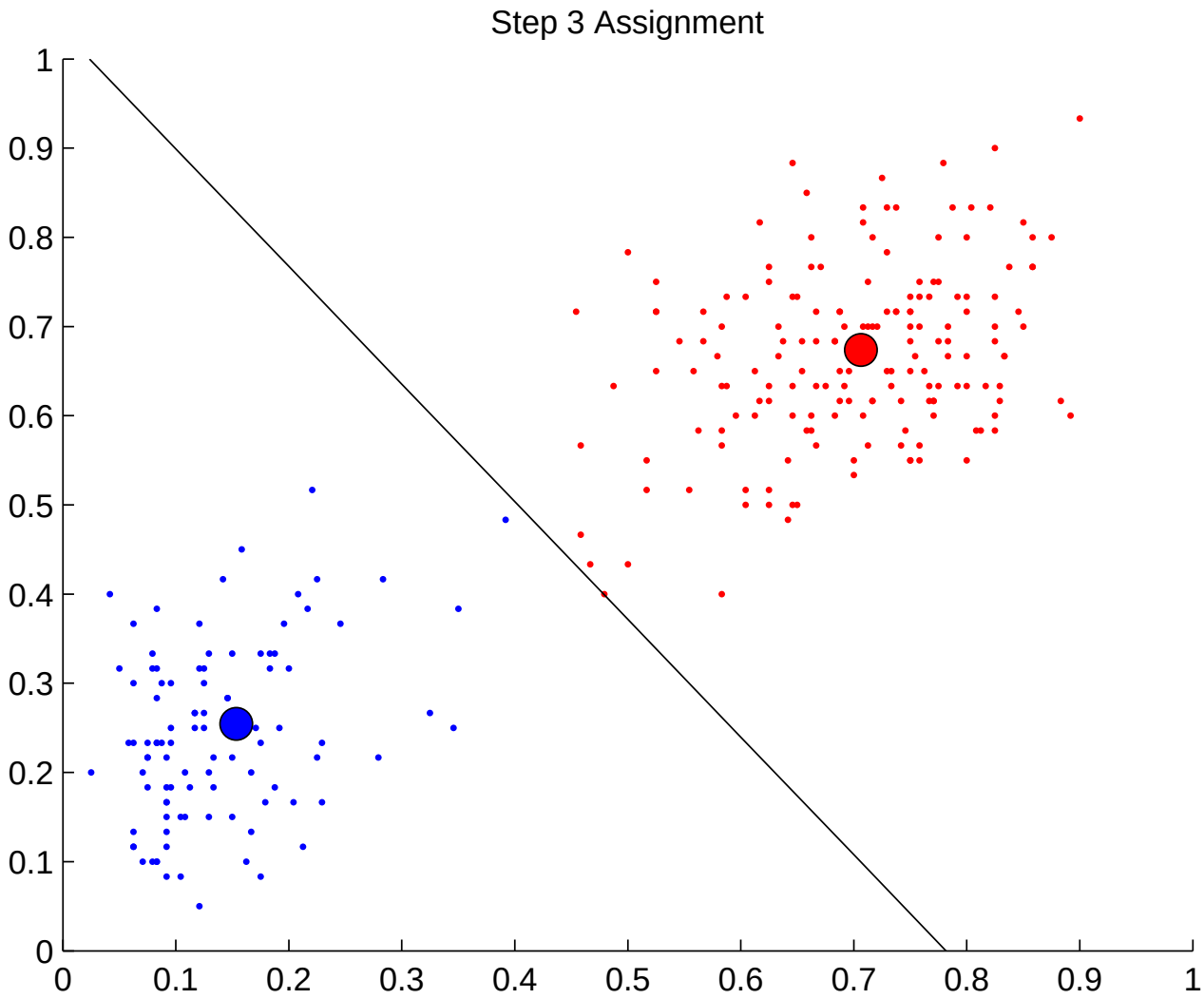
Clustering Old Faithful (cont.)



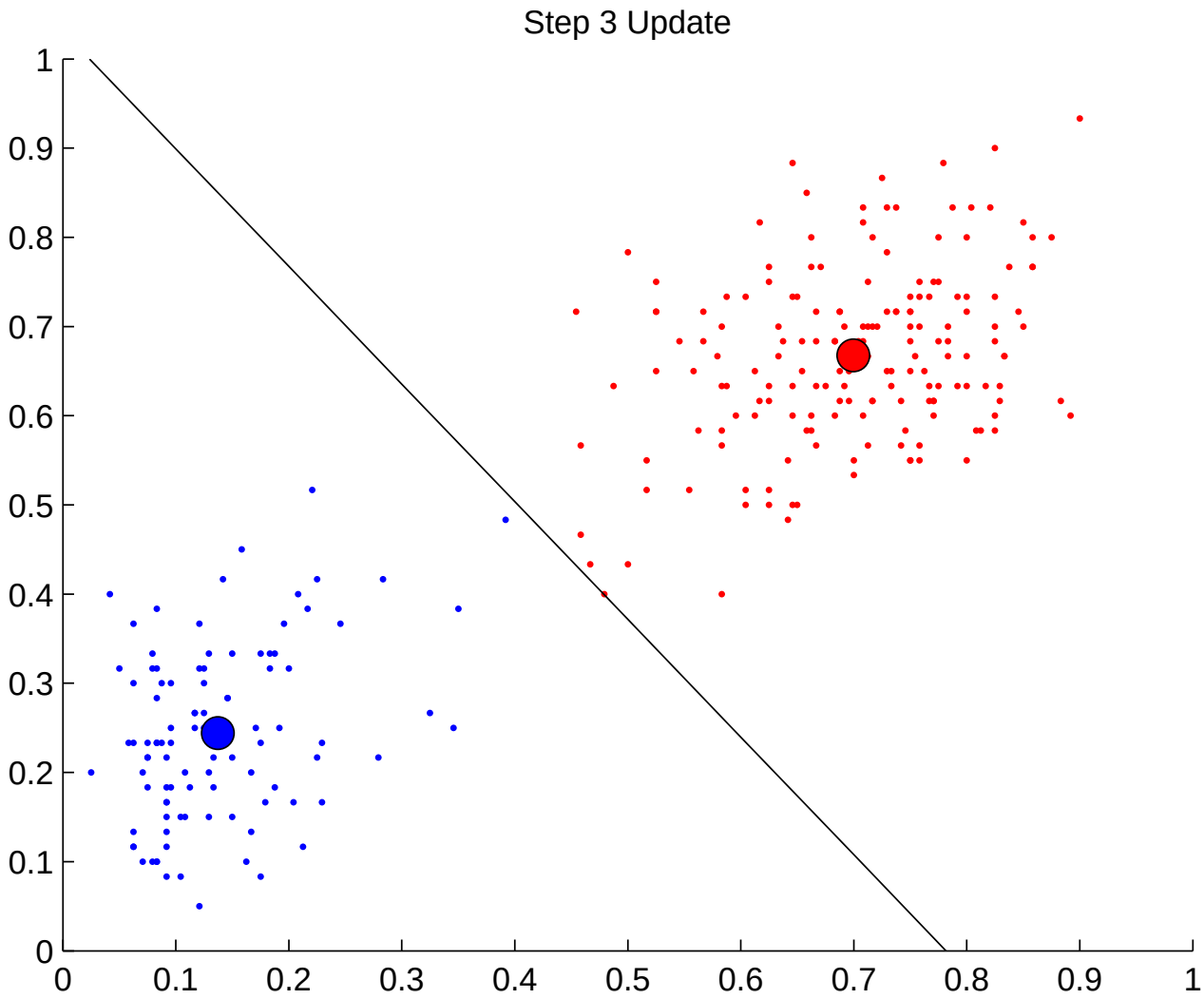
Clustering Old Faithful (cont.)



Clustering Old Faithful (cont.)

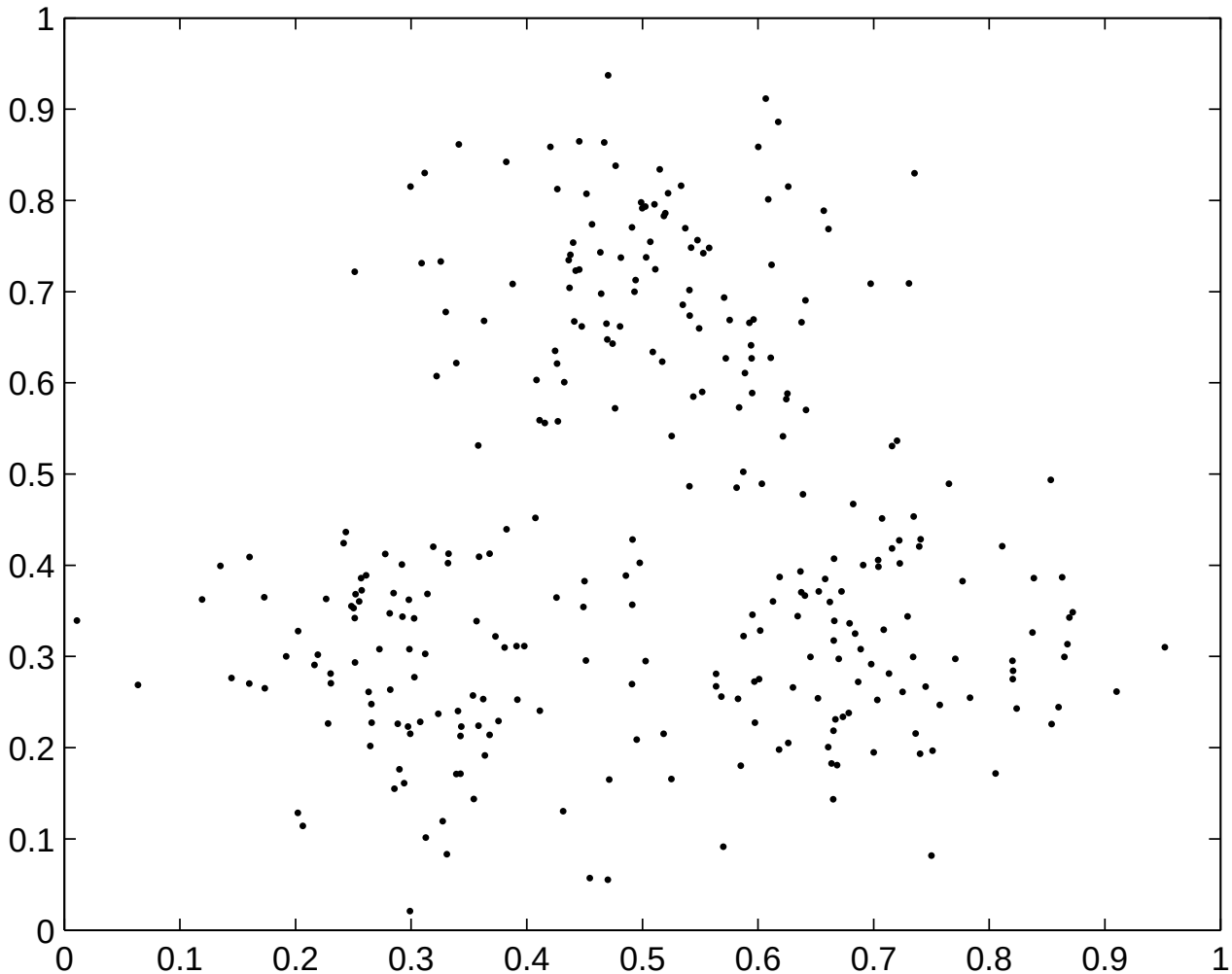


Clustering Old Faithful (cont.)

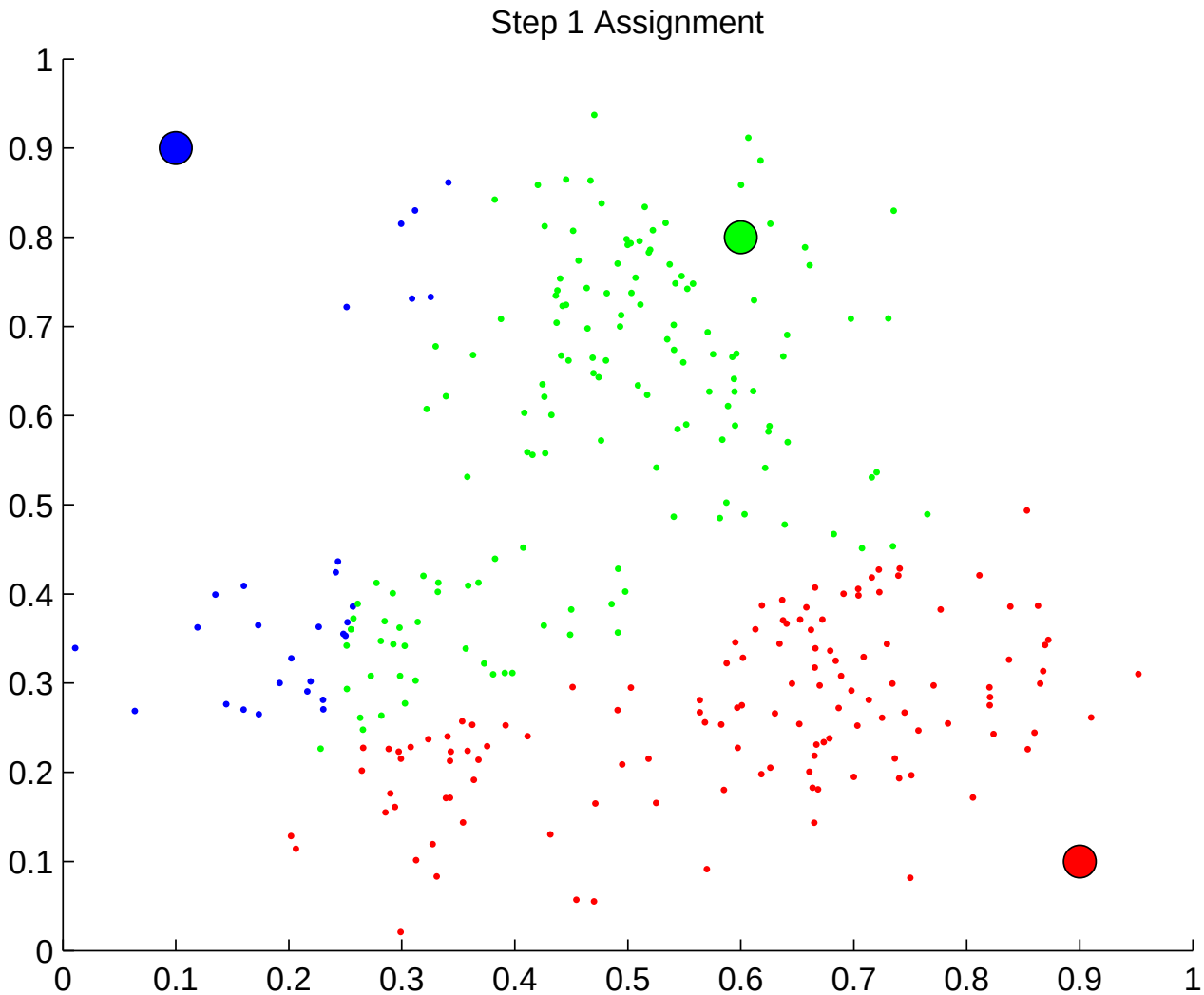


Example with 3 clusters

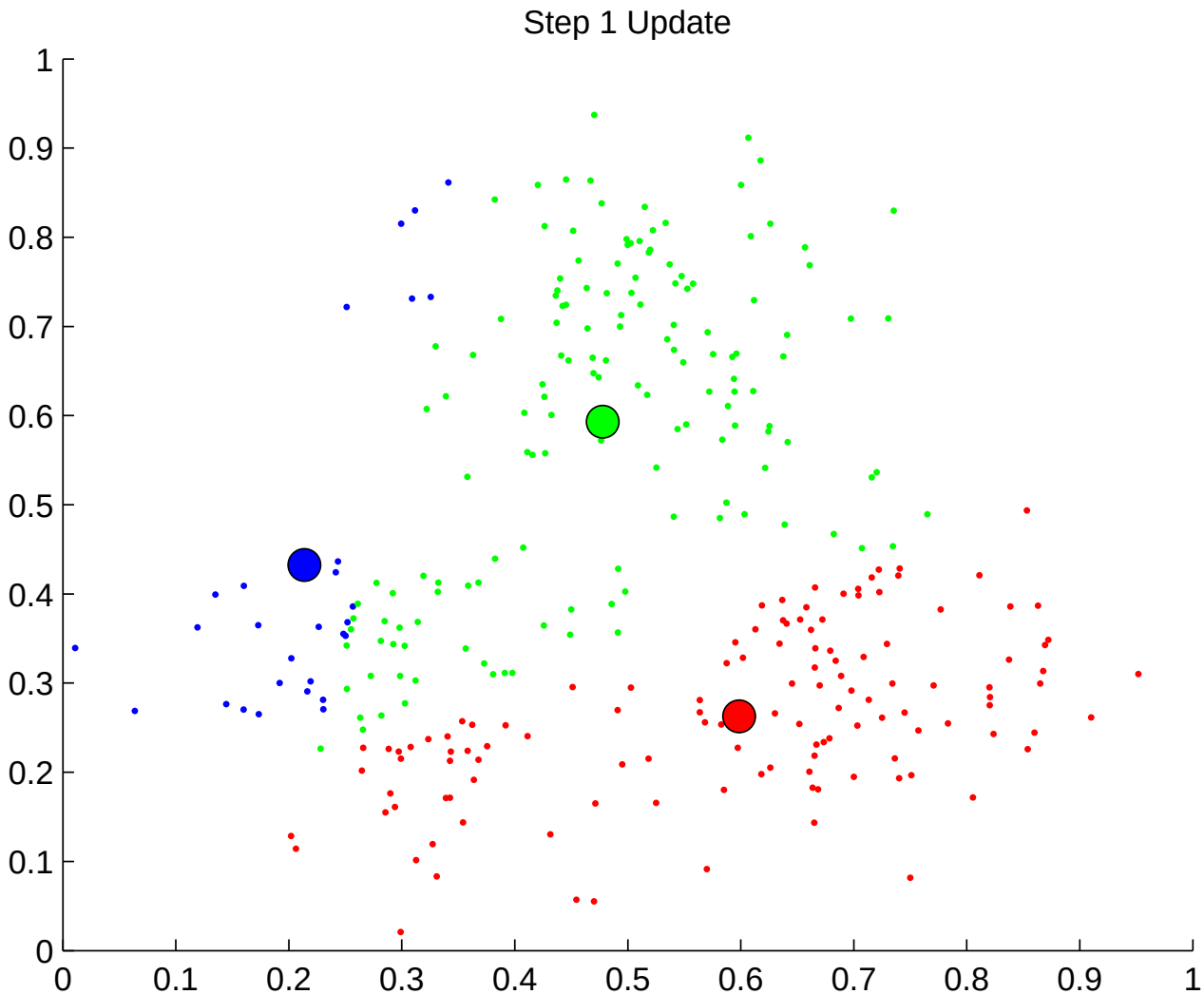
Data with roughly 3 clusters:



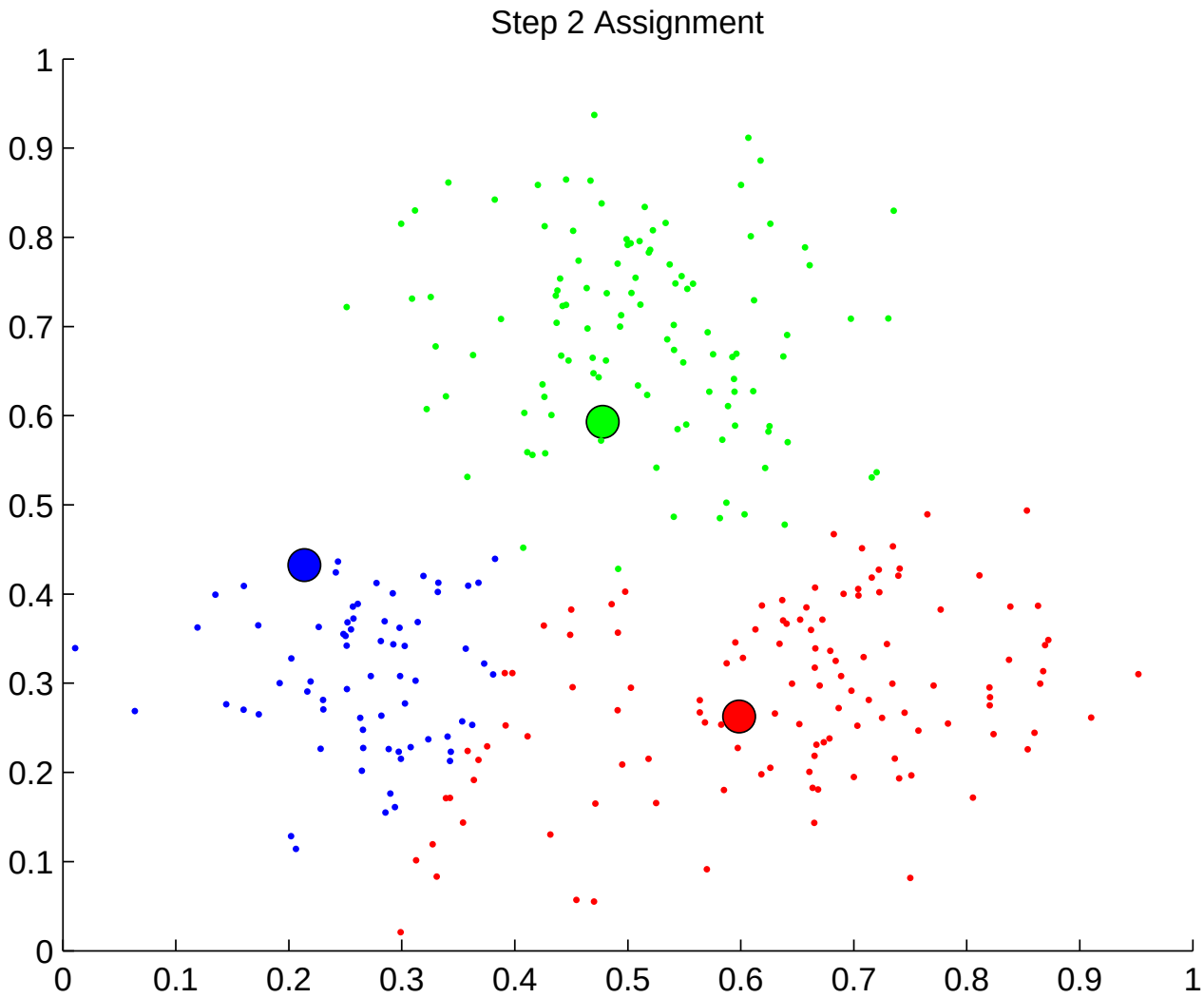
Example with 3 clusters (cont.)



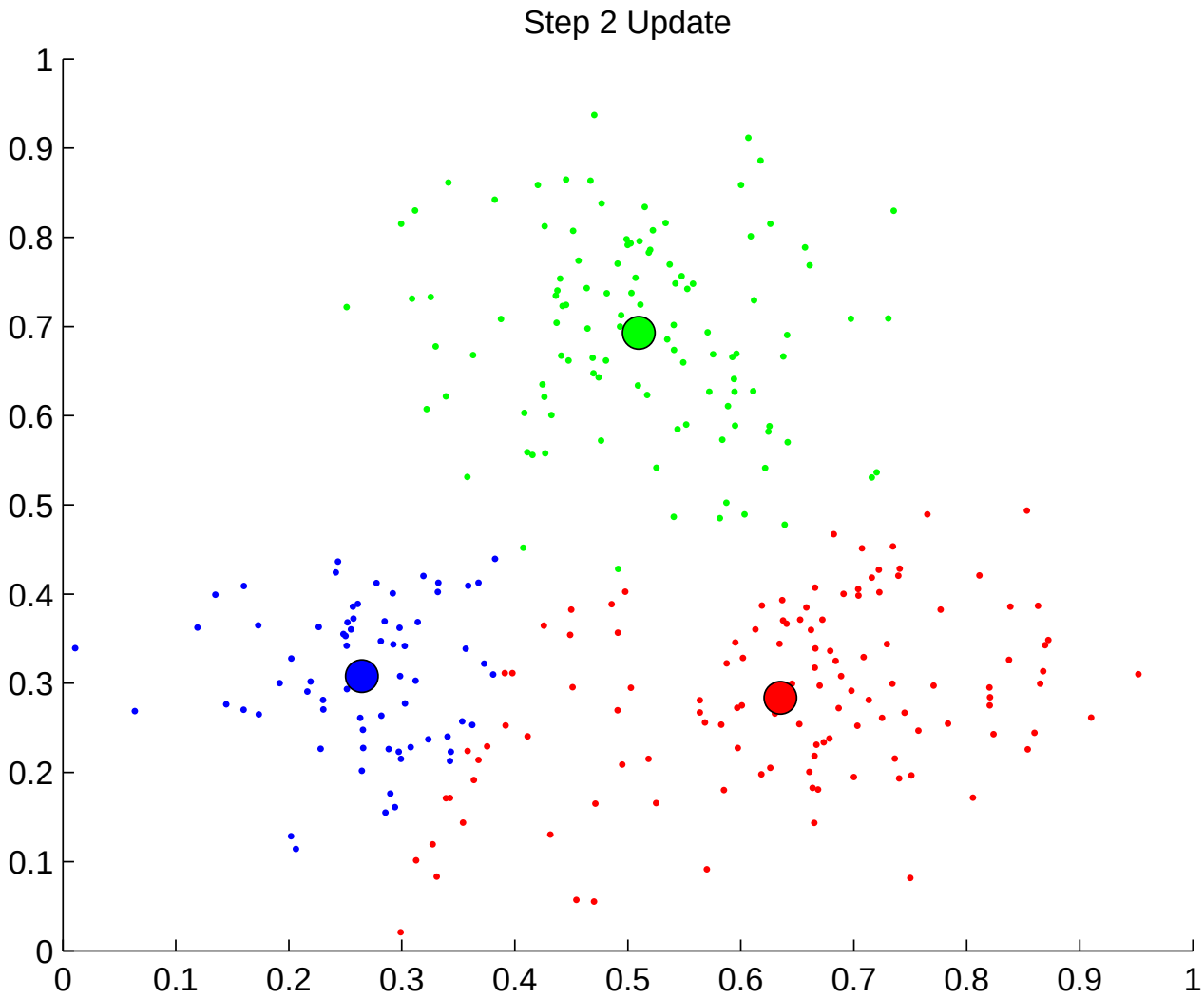
Example with 3 clusters (cont.)



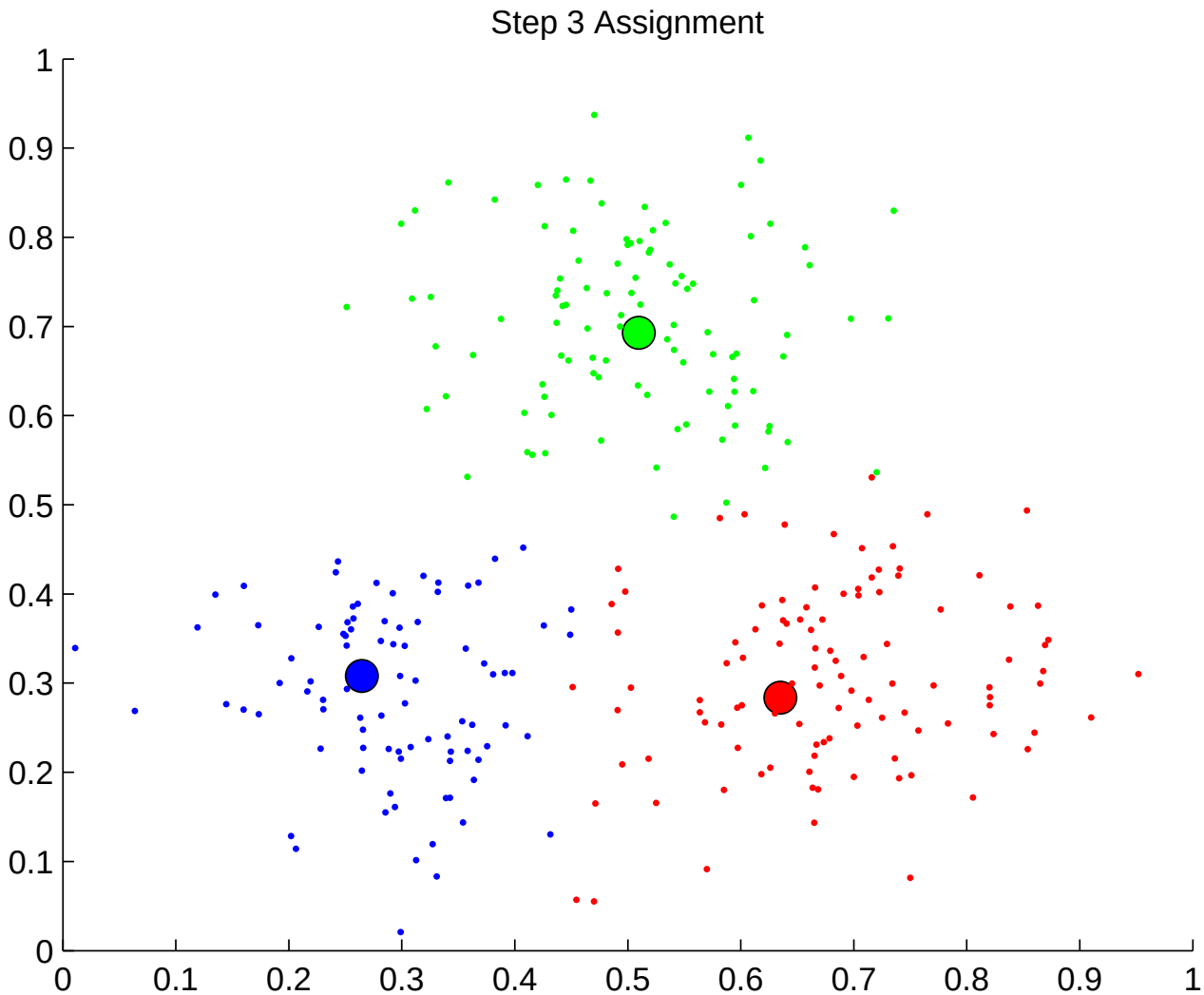
Example with 3 clusters (cont.)



Example with 3 clusters (cont.)



Example with 3 clusters (cont.)



Example with 3 clusters (cont.)

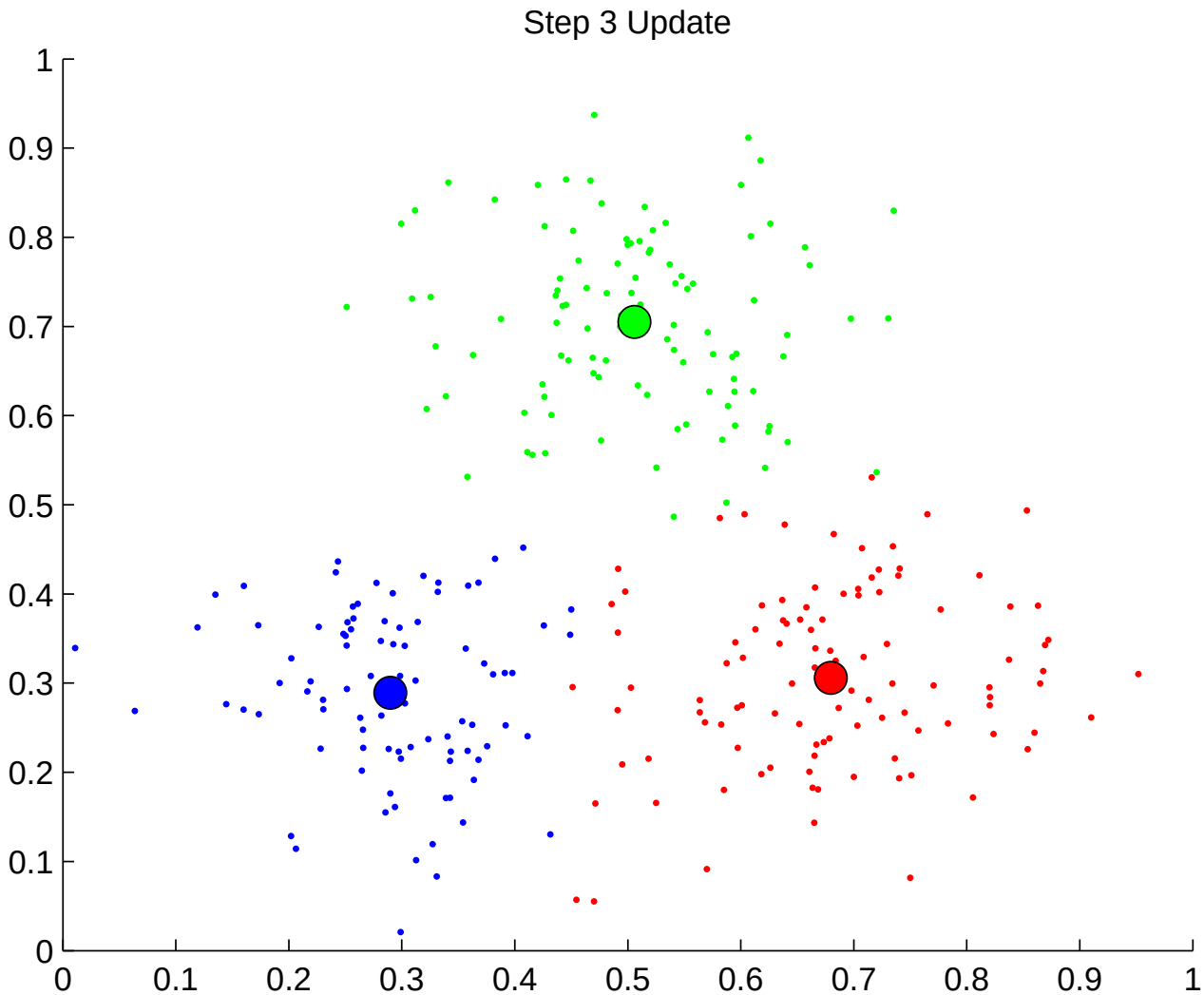
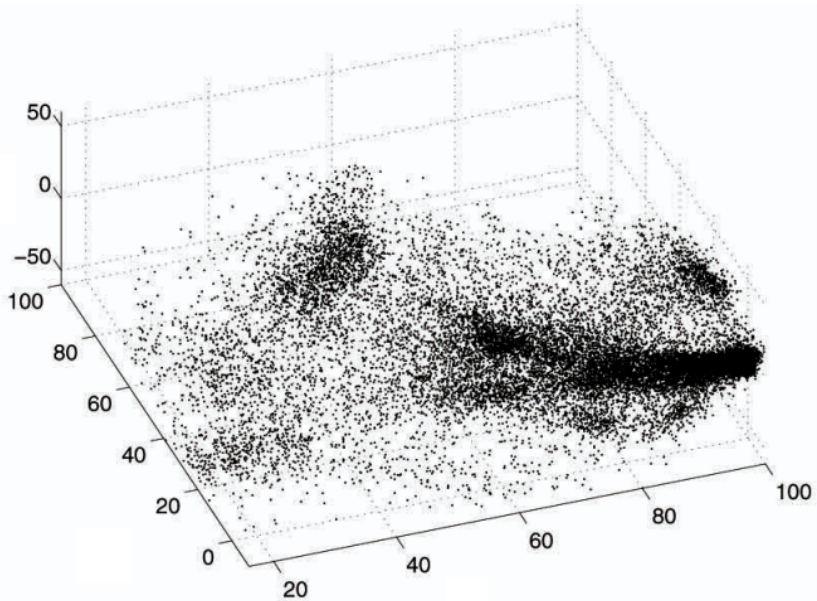


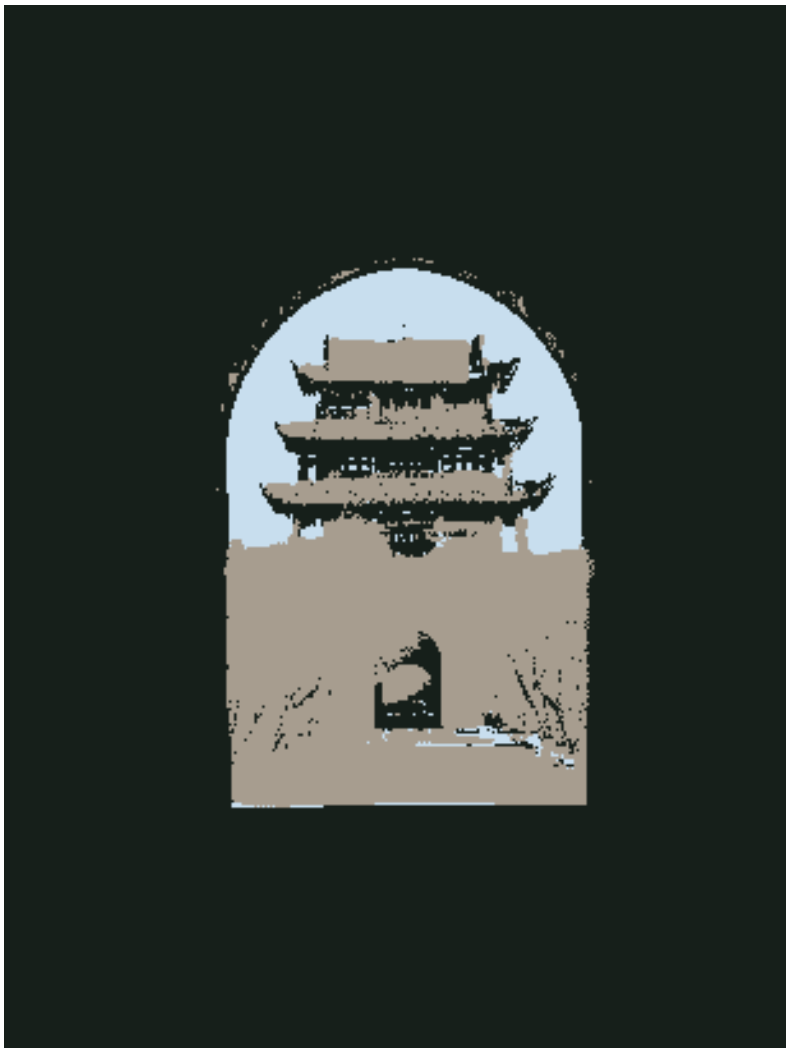
Image segmentation

In a colour digital image, each pixel has 3 separate values: Red intensity, green intensity and blue intensity. We can treat each pixel as a 3-dimensional vector $\begin{bmatrix} r & g & b \end{bmatrix}^T$.

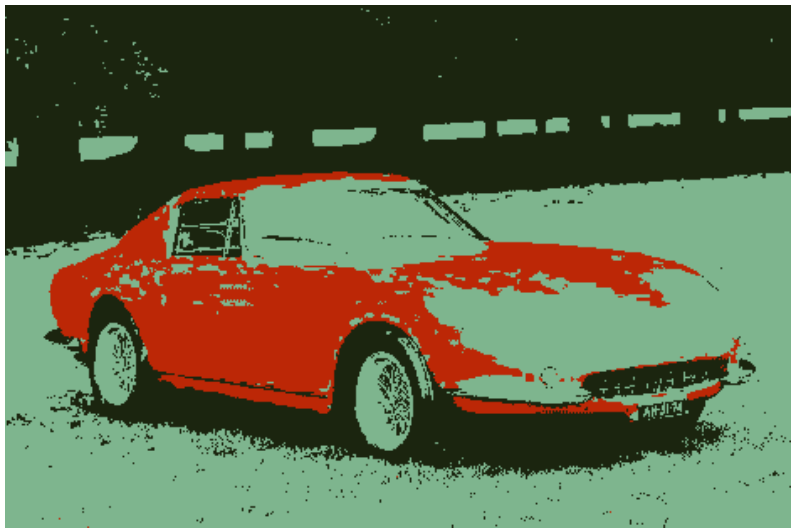


K-means with $K = 3$

K-means with $K = 3$ (cont.)



K-means with $K = 3$ (cont.)

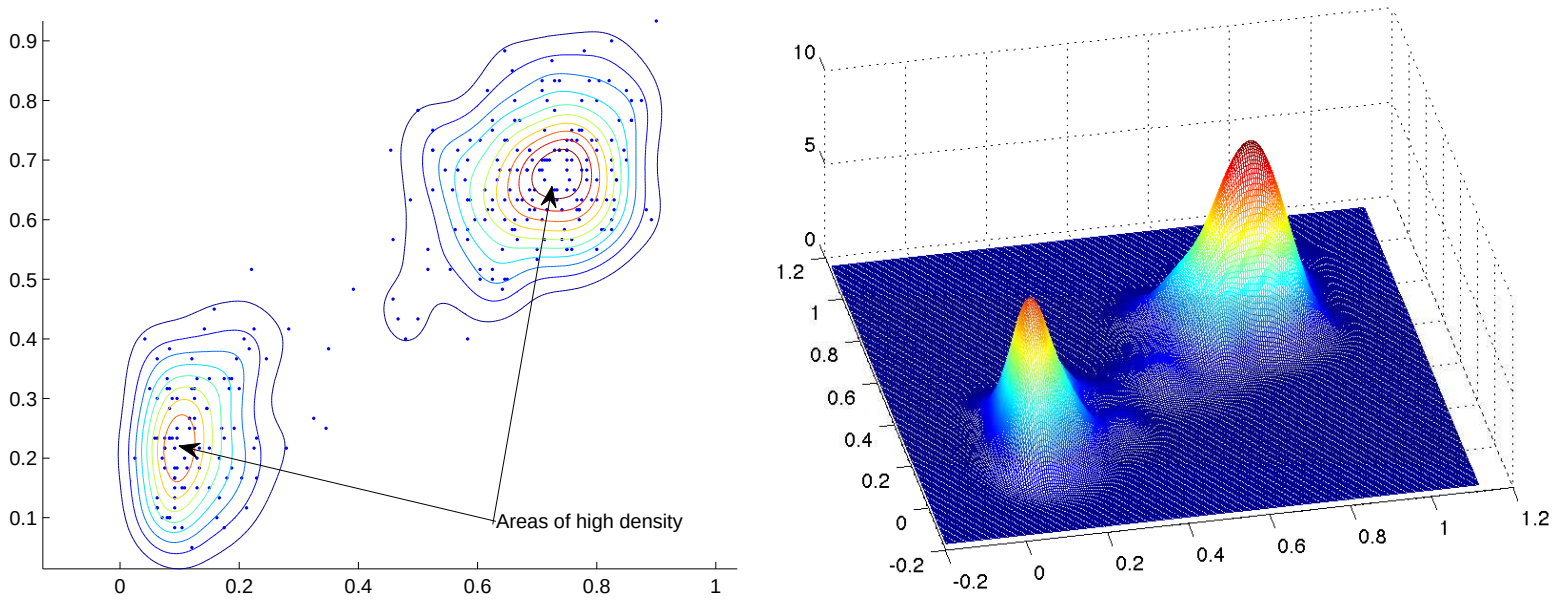


The Mean Shift Algorithm

Mean shift

Given a set of input data $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, the goal of mean shift is to **seek areas of high density** in the data.

Old Faithful dataset:



It can be seen that **peaks in the density** correspond to the centroids of clusters. Our goal is to **seek these peaks**.

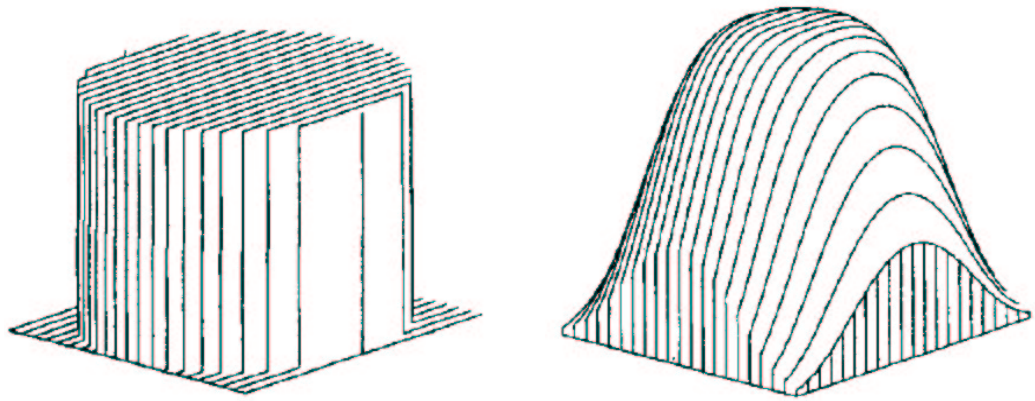
Kernel function

At the heart of the mean shift procedure is the **kernel function**.

Given an **initial guess** $\mathbf{z}$ for the centroid, the kernel function $k(\mathbf{z}, \mathbf{x})$ is applied on $\mathbf{z}$ and another point $\mathbf{x}$ in the input data.

Two popular kernels are the **uniform kernel** and **Gaussian kernel**:

$$k(\mathbf{z}, \mathbf{x}) = \begin{cases} 1 & \text{if } \|\mathbf{z} - \mathbf{x}\| \leq h \\ 0 & \text{otherwise} \end{cases} \qquad k(\mathbf{z}, \mathbf{x}) = \exp^{-\frac{\|\mathbf{z}-\mathbf{x}\|^2}{2h^2}}$$



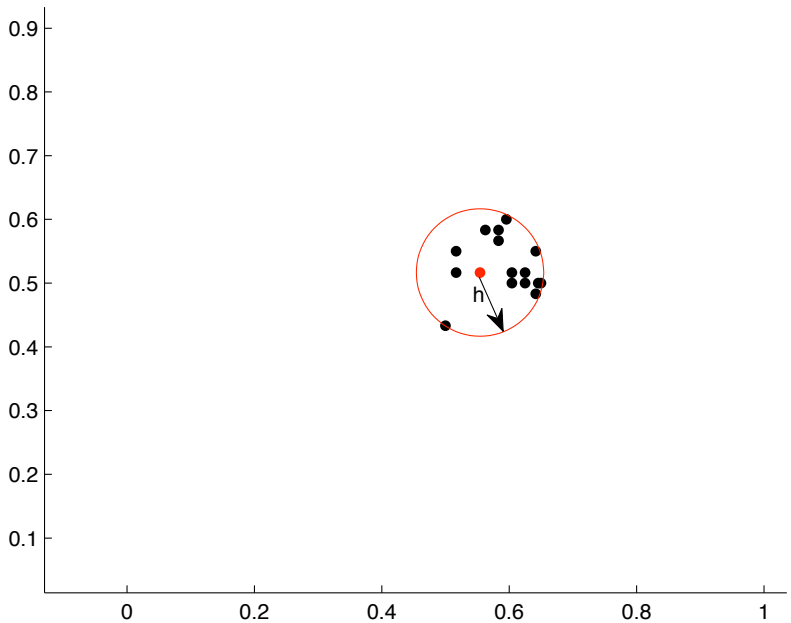
The value $\|\mathbf{z} - \mathbf{x}\|$ is simply the Euclidean distance between $\mathbf{z}$ and $\mathbf{x}$. Thus the **closer** $\mathbf{z}$ and $\mathbf{x}$ are, the **higher** the kernel response.

The kernels depend on parameter h called the **bandwidth**.

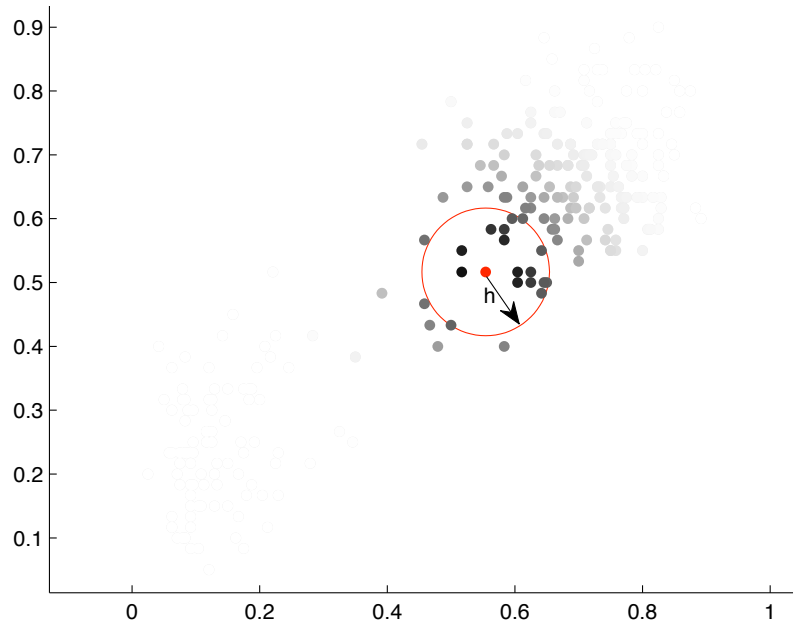
Bandwidth

If we fix $\mathbf{z}$ and let $\mathbf{x}$ vary, the bandwidth specifies the size of the **neighbourhood** around $\mathbf{z}$ according to $k(\mathbf{z}, \mathbf{x})$:

Uniform (red point is $\mathbf{z}$):



Gaussian (red point is $\mathbf{z}$):



In the above figures, the **darkness** of a point $\mathbf{x}_i$ is proportional to the kernel response $k(\mathbf{z}, \mathbf{x}_i)$.

The uniform kernel provides a "hard" boundary while the Gaussian kernel which provides a "soft" boundary.

The mean shift update

The **mean shift update** causes a point $\mathbf{z}$ to be revised to become the mean vector of its neighbourhood. Mathematically,

$$\mathbf{z}_{new} = \frac{\sum_{i=1}^{N} k(\mathbf{z}, \mathbf{x}_i)\mathbf{x}_i}{\sum_{i=1}^{N} k(\mathbf{z}, \mathbf{x}_i)}$$

In the numerator, each $k(\mathbf{z}, \mathbf{x}_i)\mathbf{x}_i$ implies a **weighting** of point $\mathbf{x}_i$ according to the closeness of $\mathbf{x}_i$ to $\mathbf{z}$. The closer $\mathbf{x}_i$ is to $\mathbf{z}$, the higher the kernel output $k(\mathbf{z}, \mathbf{x}_i)$ and the larger the effect $\mathbf{x}_i$ has on computing $\mathbf{z}_{new}$.

The denomenator simply provides normalisation constant such that the weights sum to 1.

Example

Let $\mathbf{z} = \begin{bmatrix} 0.55 \\ 0.52 \end{bmatrix}$ and

$$X = \left\{ \begin{bmatrix} 0.58 \\ 0.57 \end{bmatrix}, \begin{bmatrix} 0.14 \\ 0.28 \end{bmatrix}, \begin{bmatrix} 0.73 \\ 0.65 \end{bmatrix}, \begin{bmatrix} 0.63 \\ 0.52 \end{bmatrix}, \begin{bmatrix} 0.50 \\ 0.45 \end{bmatrix}, \begin{bmatrix} 0.84 \\ 0.77 \end{bmatrix} \right\}$$

The distance between $\mathbf{z}$ and the other points are respectively 0.0583, 0.4751, 0.2220, 0.0800, 0.0860 and 0.3829.

Using the **uniform kernel** with bandwidth $h = 0.1$, the kernel responses between $\mathbf{z}$ and each input point are

$$k(\mathbf{z}, \mathbf{x}_1) = k(\mathbf{z}, \mathbf{x}_4) = k(\mathbf{z}, \mathbf{x}_5) = 1$$
$$k(\mathbf{z}, \mathbf{x}_2) = k(\mathbf{z}, \mathbf{x}_3) = k(\mathbf{z}, \mathbf{x}_6) = 0$$

Thus $\mathbf{z}$ is updated to become

$$\mathbf{z}_{new} = \frac{k(\mathbf{z}, \mathbf{x}_1)\mathbf{x}_1 + k(\mathbf{z}, \mathbf{x}_4)\mathbf{x}_4 + k(\mathbf{z}, \mathbf{x}_5)\mathbf{x}_5}{k(\mathbf{z}, \mathbf{x}_1) + k(\mathbf{z}, \mathbf{x}_4) + k(\mathbf{z}, \mathbf{x}_5)} = \begin{bmatrix} 0.55 \\ 0.52 \end{bmatrix}$$

Example (cont.)

Using the **Gaussian kernel** with $h = 0.1$, the kernel responses are

$$k(\mathbf{z}, \mathbf{x}_1) = 0.8437,\ k(\mathbf{z}, \mathbf{x}_2) = 0.0000,\ k(\mathbf{z}, \mathbf{x}_3) = 0.0850,$$
$$k(\mathbf{z}, \mathbf{x}_4) = 0.7261,\ k(\mathbf{z}, \mathbf{x}_5) = 0.6907,\ k(\mathbf{z}, \mathbf{x}_6) = 0.0007$$

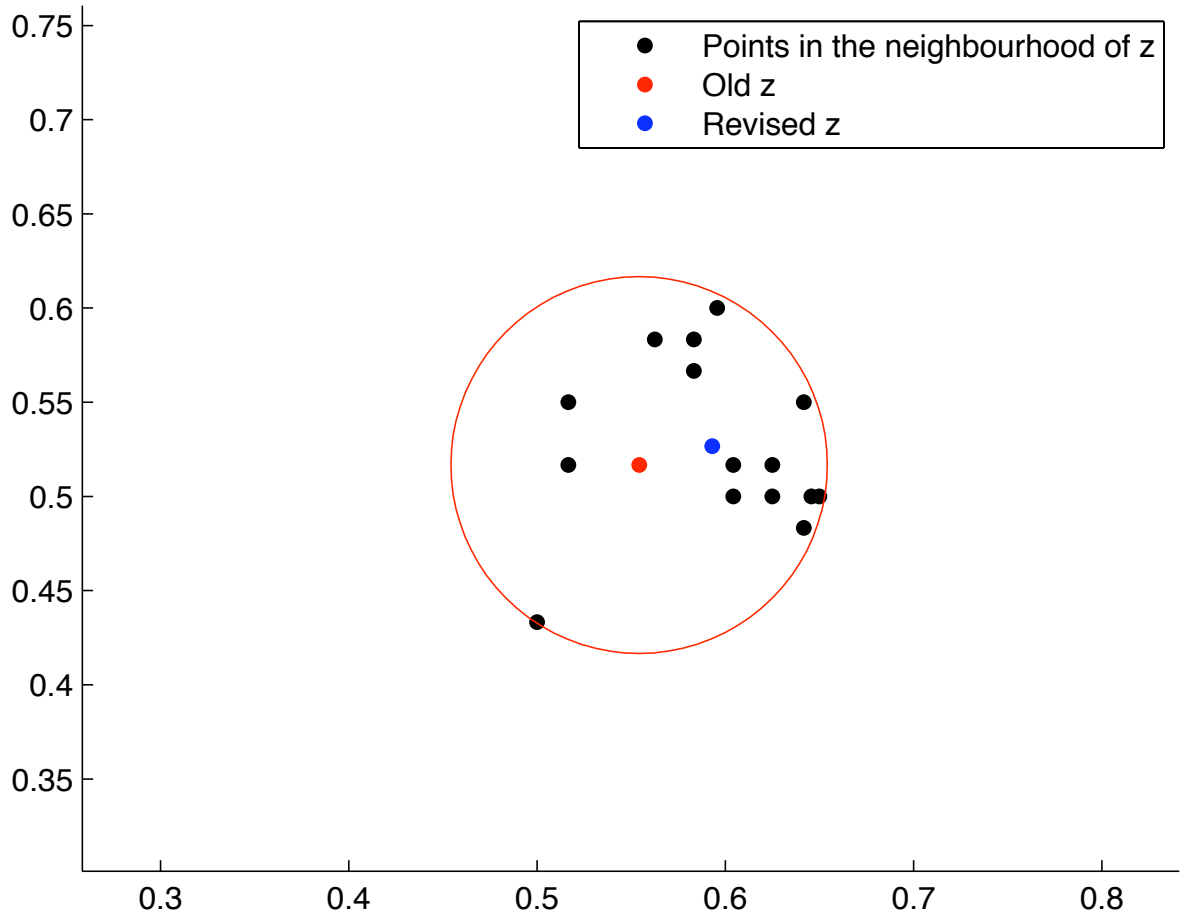
Thus $\mathbf{z}$ is updated to become

$$\mathbf{z}_{new} = \frac{0.8437\mathbf{x}_1 + 0.0000\mathbf{x}_2 + 0.0850\mathbf{x}_3 + 0.7261\mathbf{x}_4 + 0.6907\mathbf{x}_5 + 0.0007\mathbf{x}_6}{0.8437 + 0.0000 + 0.0850 + 0.7261 + 0.6907 + 0.0007}$$
$$= \begin{bmatrix} 0.5774 \\ 0.5221 \end{bmatrix}$$

Observe that for the Gaussian kernel, points that are outside the neighbourhood circle of $\mathbf{z}$ are still taken into account, although to a lesser degree.

Mean shift update example with the uniform kernel

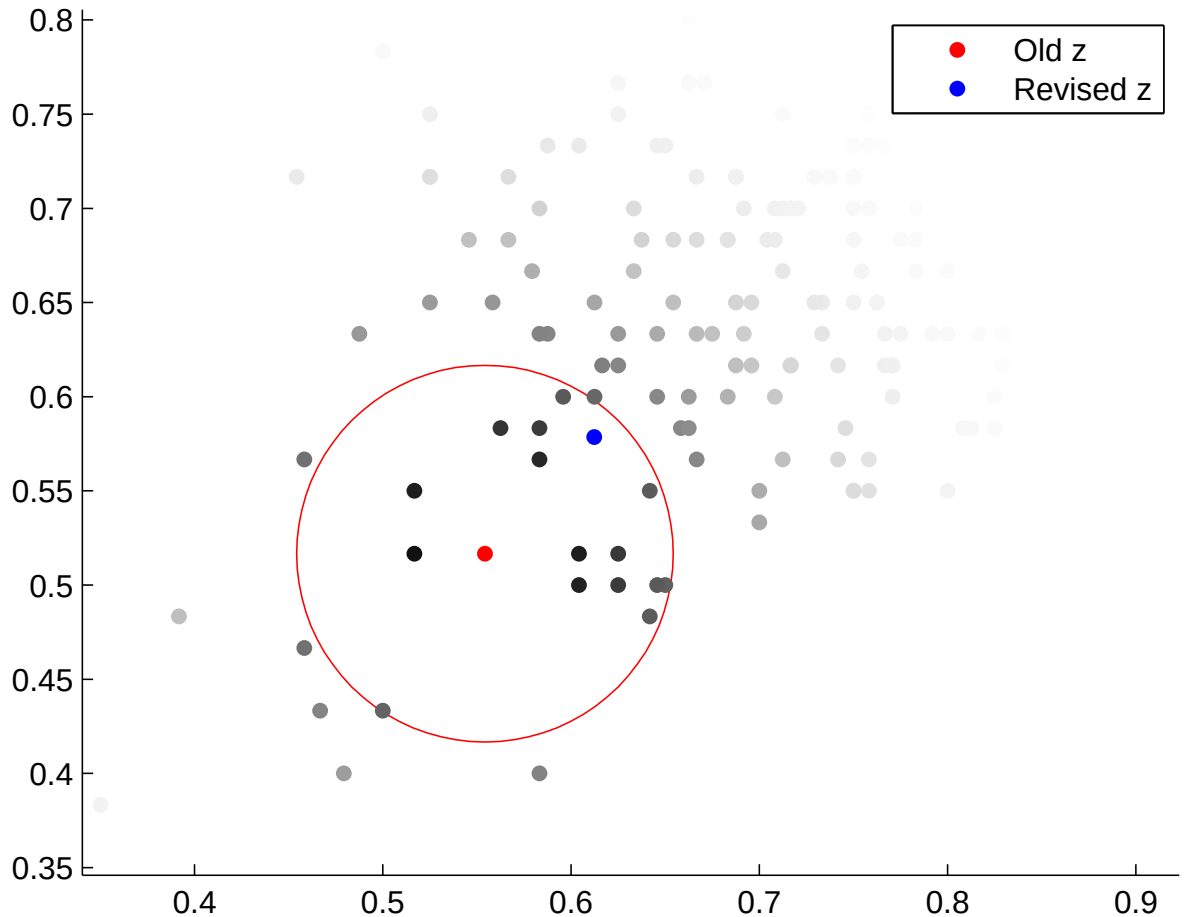
Observe that the initial guess of the cluster centre $\mathbf{z}$ has **moved towards an area with a higher density**:



In other words, $\mathbf{z}$ has **shifted towards the mean** of its neighbourhood.

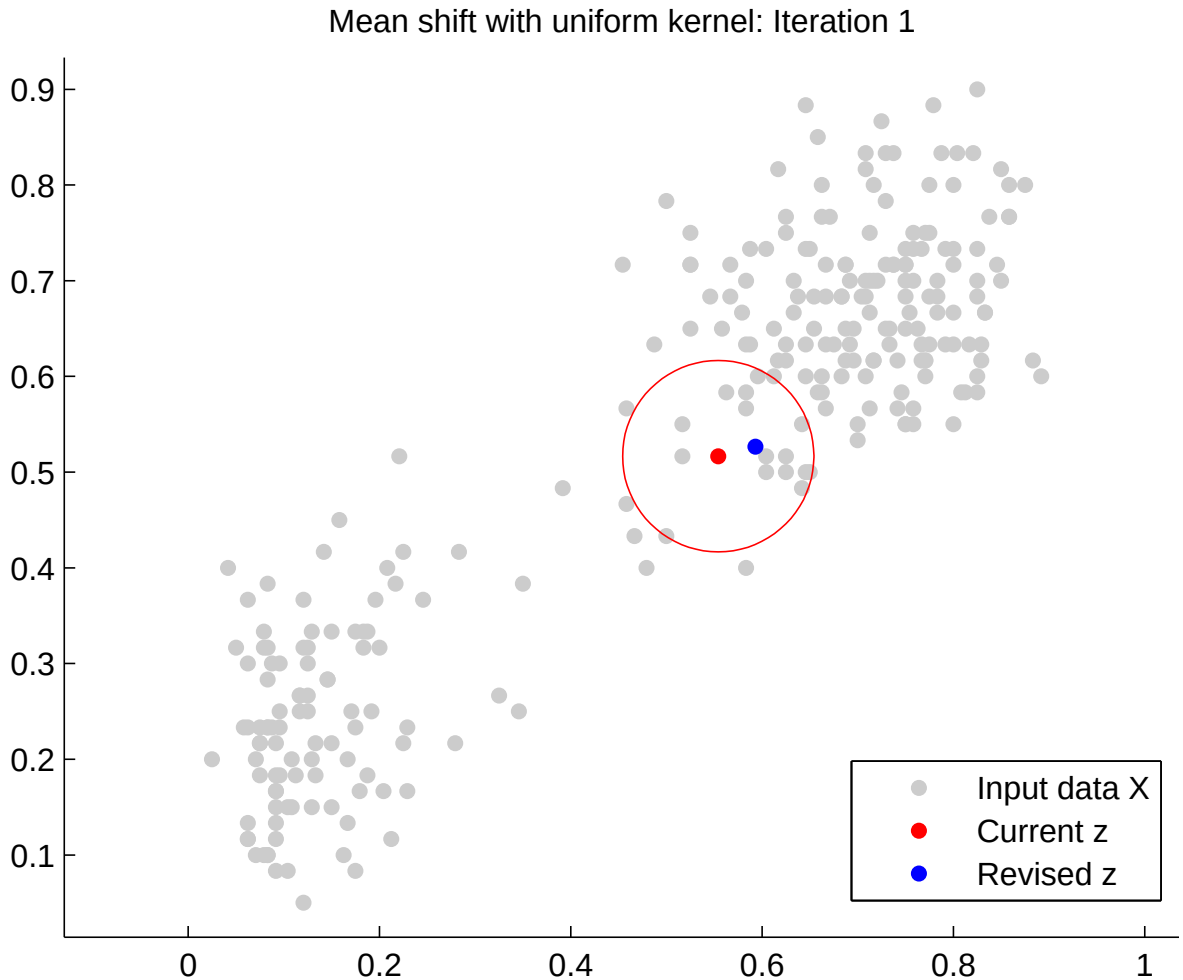
Mean shift update example with Gaussian kernel

The same behaviour can be observed for the Gaussian kernel case:



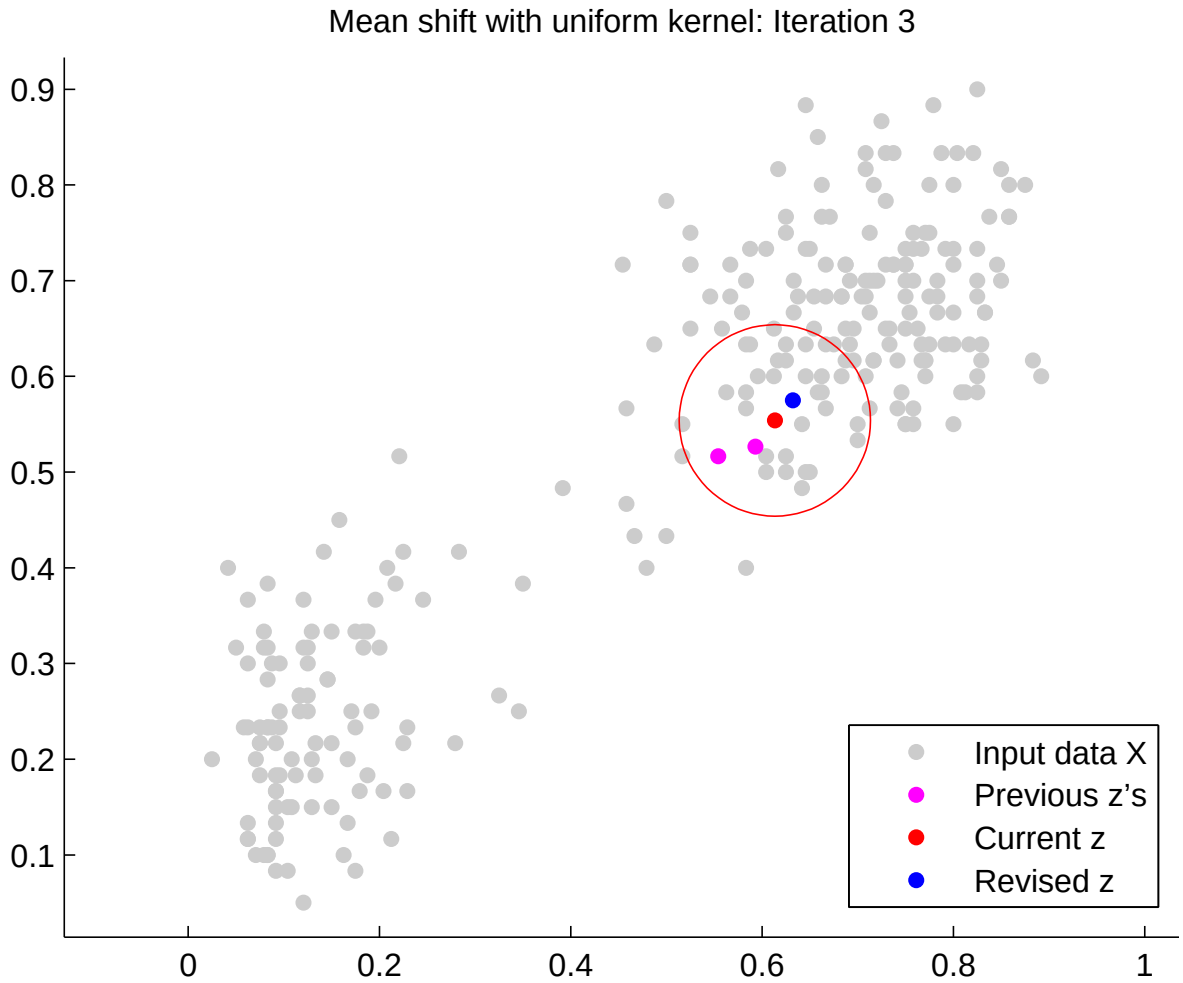
Seeking clusters with mean shift

We randomly initialise a position for a cluster centre (e.g. choose a point randomly from the dataset). Iterate the mean shift procedure on the initial position until convergence.

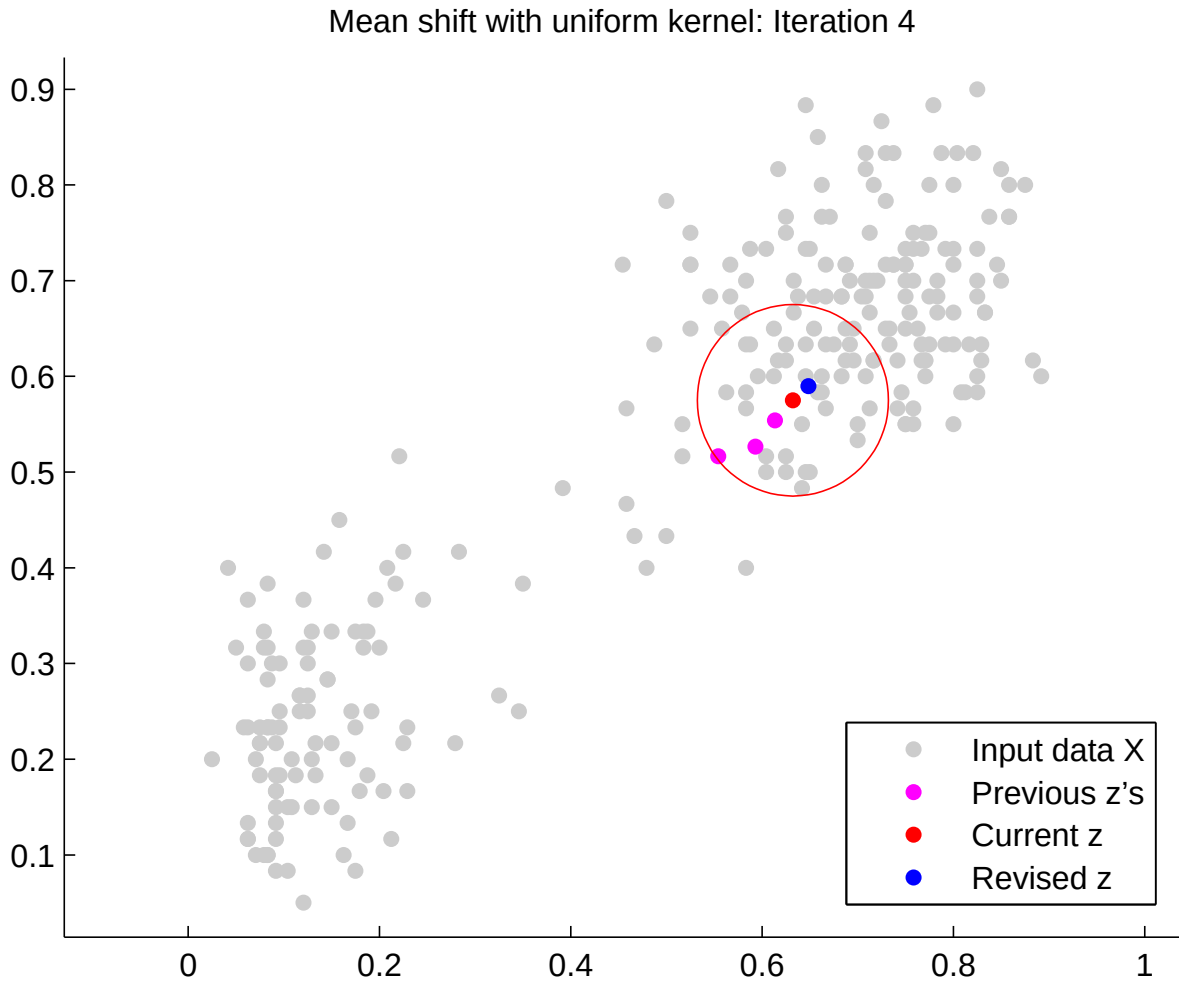


Seeking clusters with mean shift (cont.)

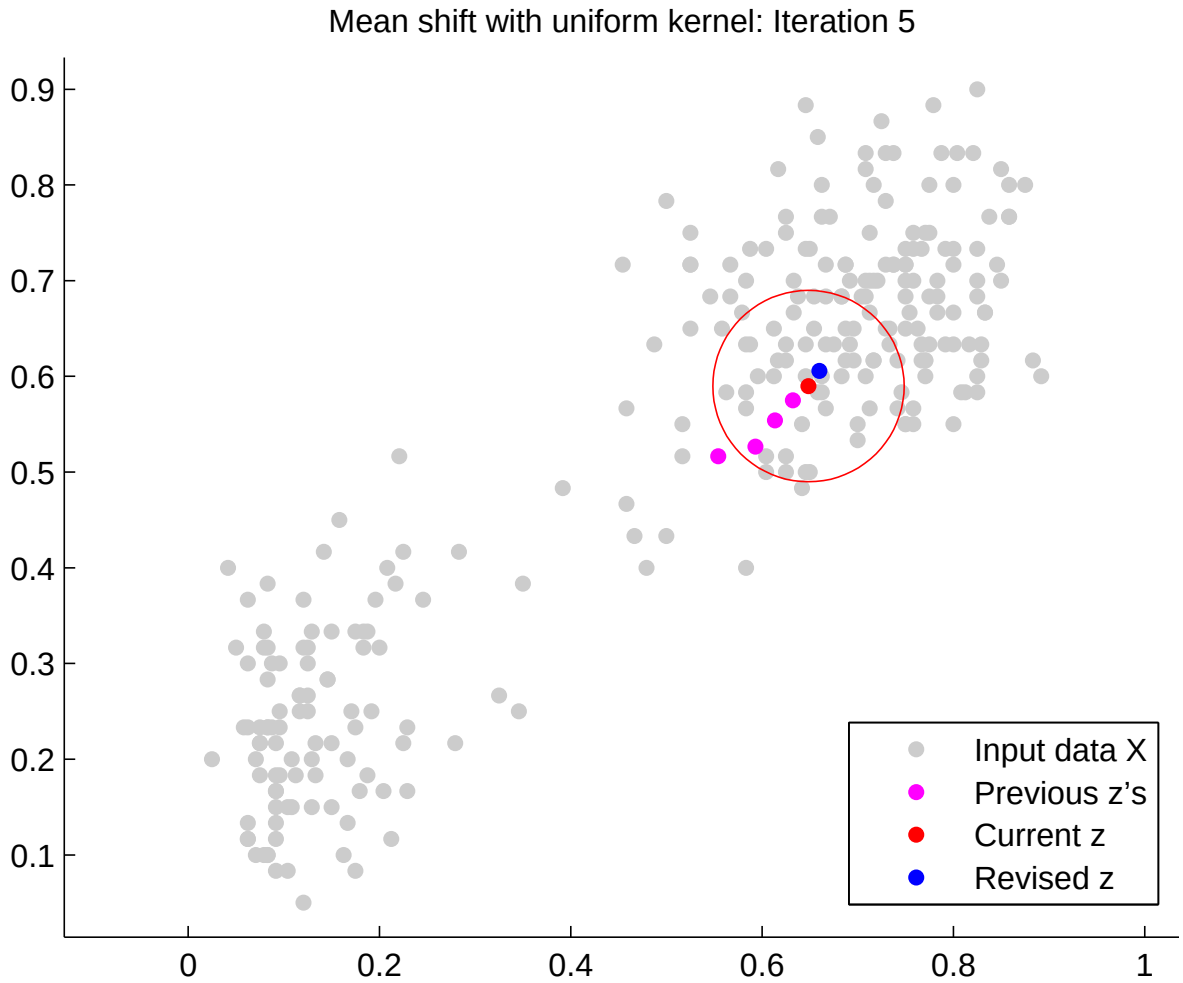
Seeking clusters with mean shift (cont.)



Seeking clusters with mean shift (cont.)

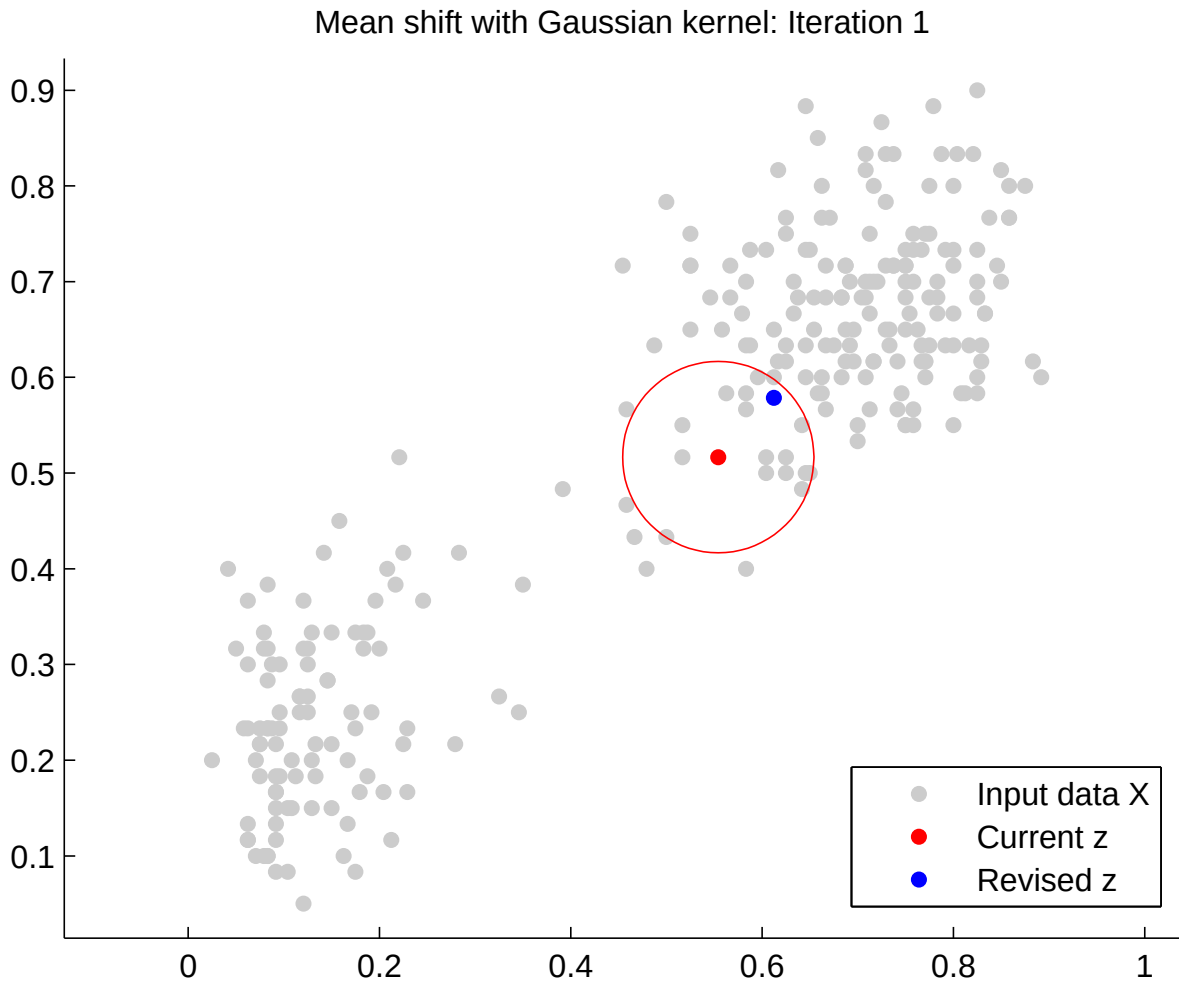


Seeking clusters with mean shift (cont.)

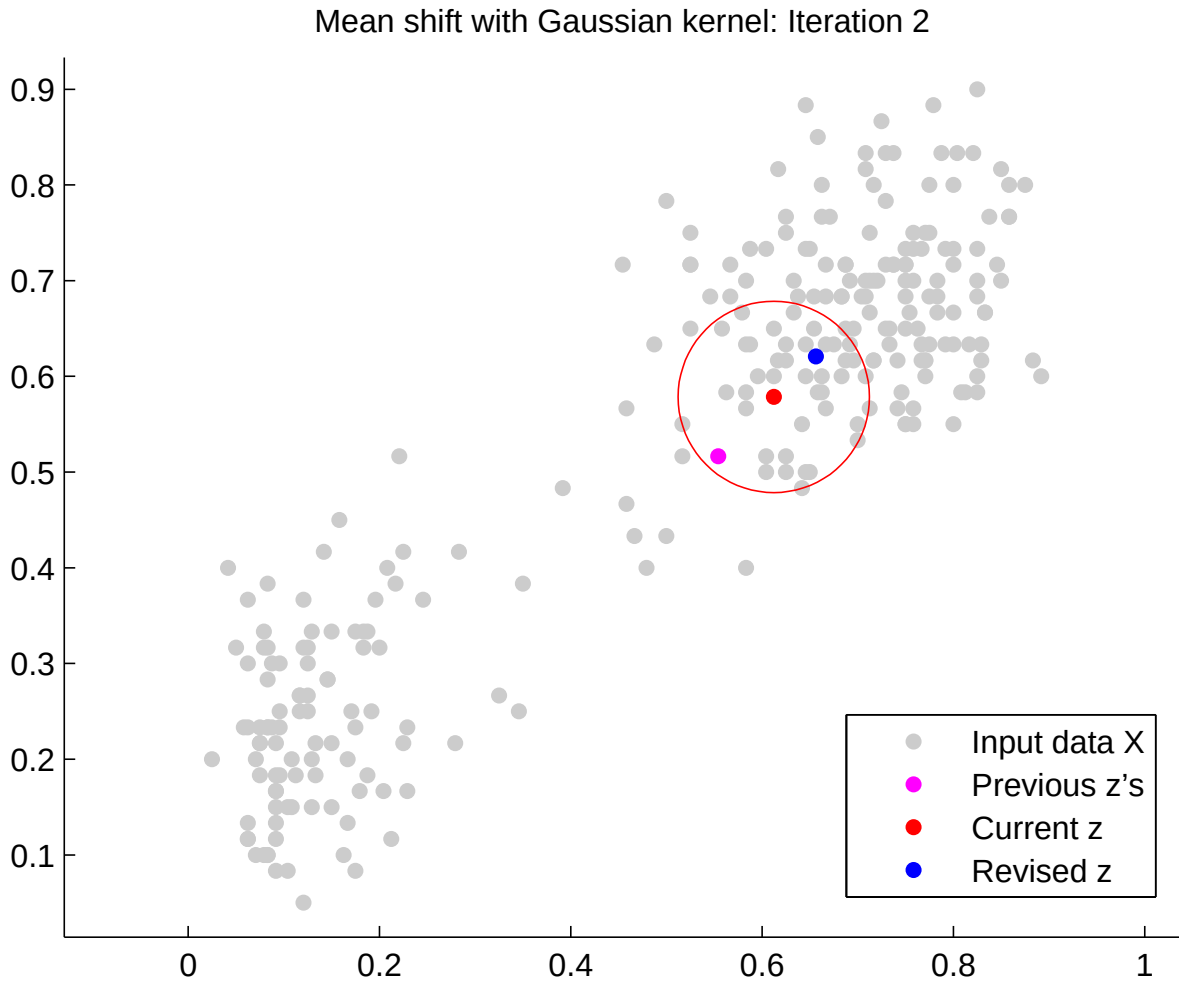


Seeking clusters with mean shift (cont.)

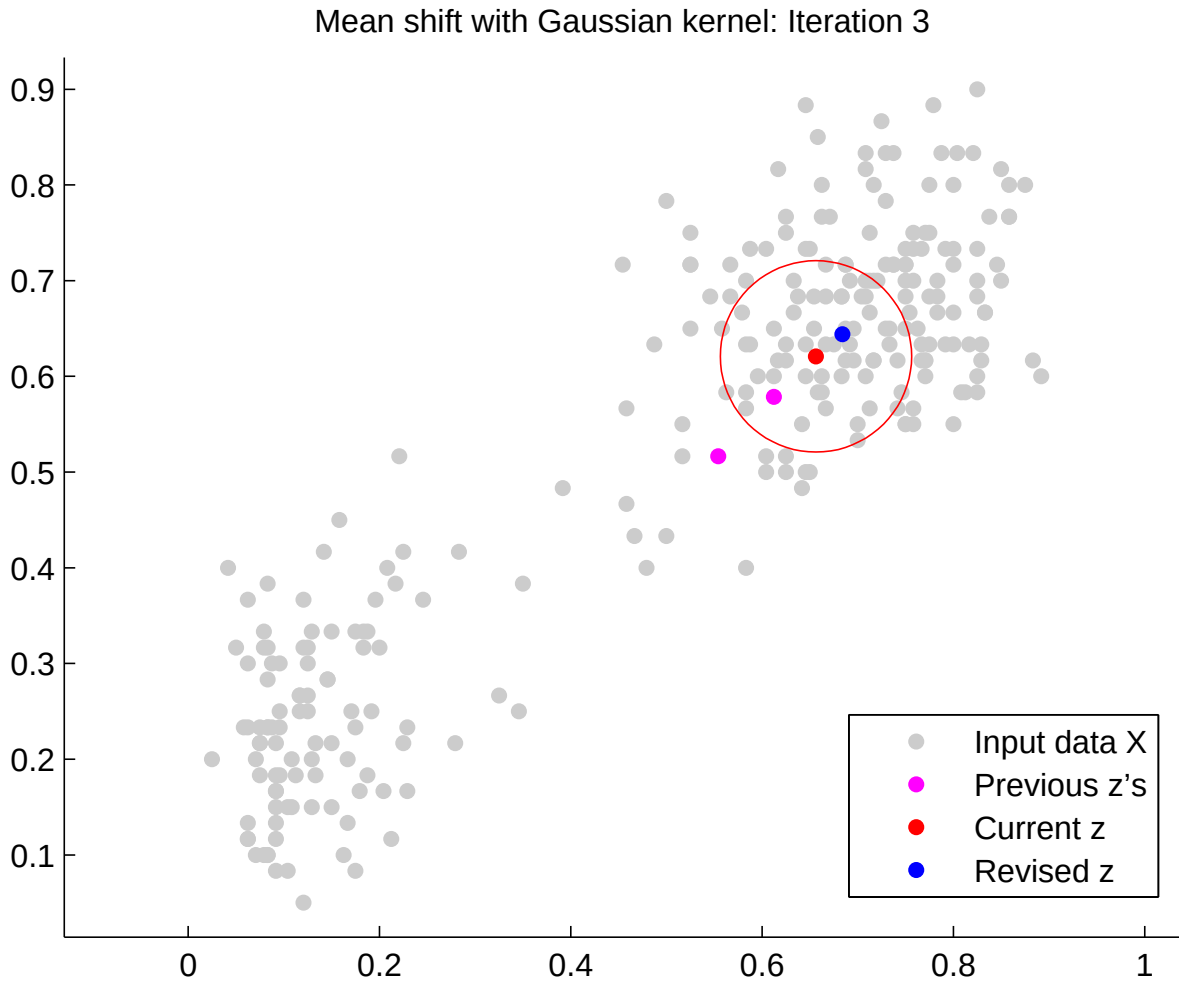
Running on the same data with the Gaussian kernel:



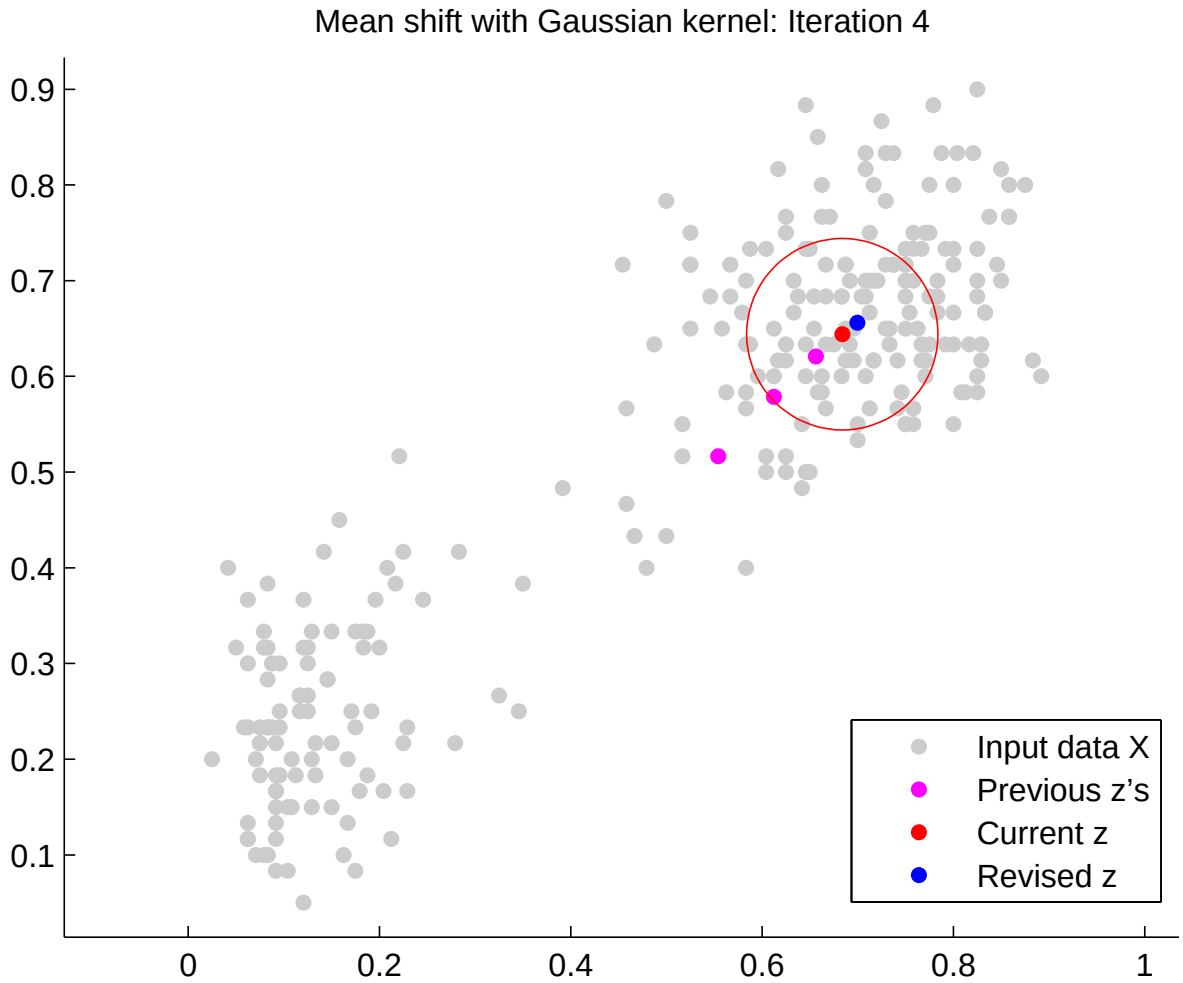
Seeking clusters with mean shift (cont.)



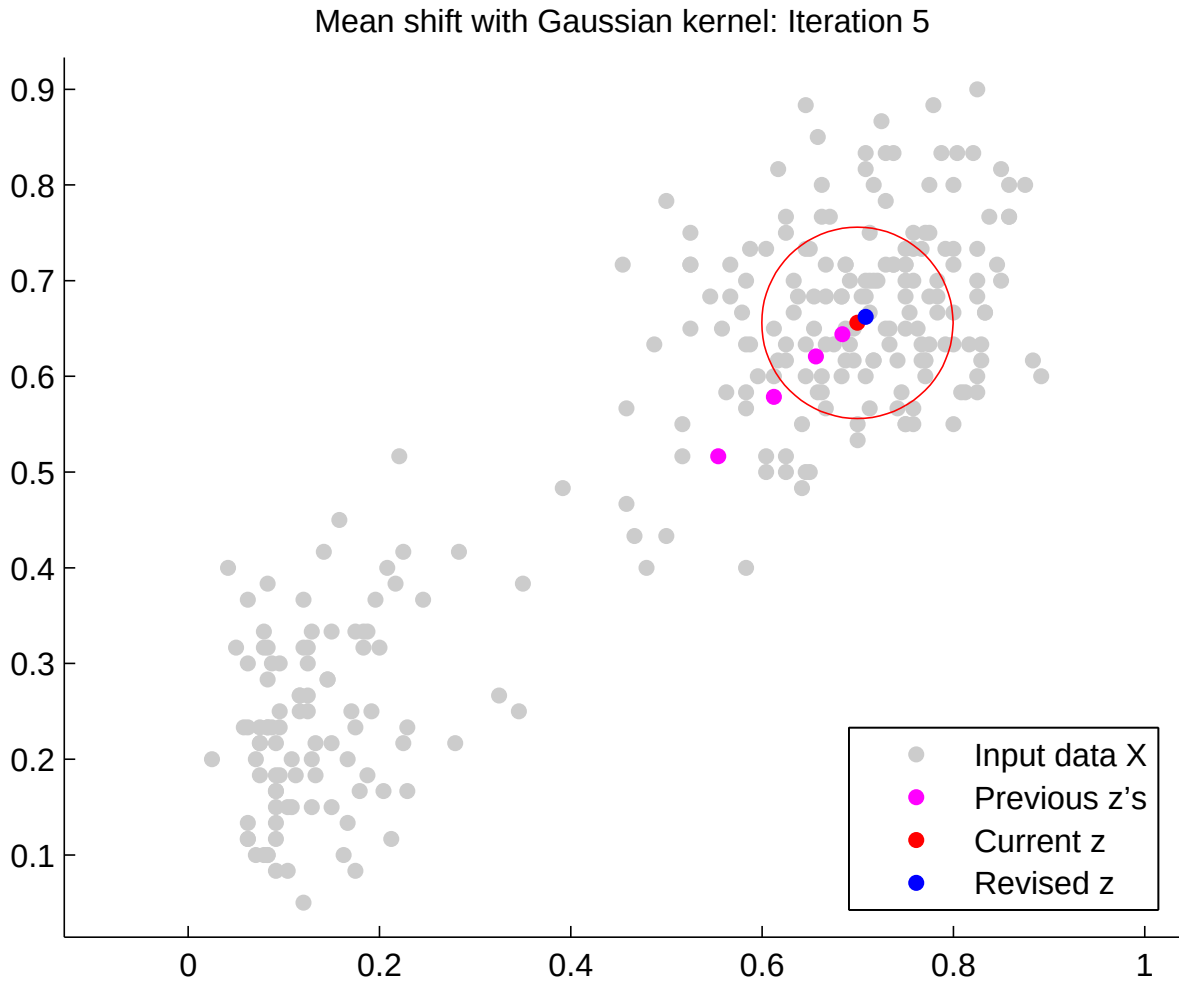
Seeking clusters with mean shift (cont.)



Seeking clusters with mean shift (cont.)



Seeking clusters with mean shift (cont.)



Mean shift algorithm

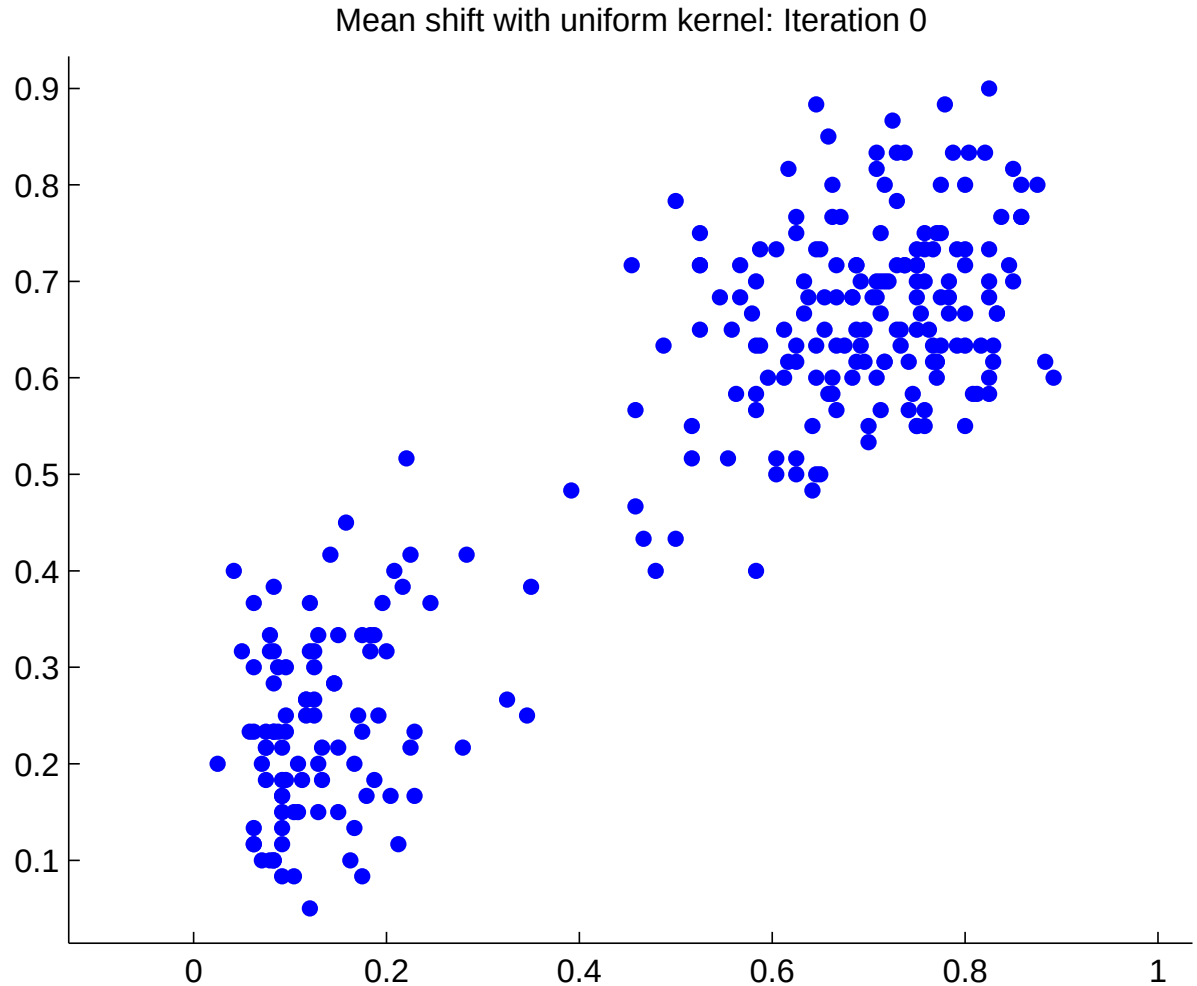
In most practical applications all input points are used as the initial estimates for the cluster centres. The mean shift procedure is then applied on all the initial estimates until convergence, e.g. when all the revised estimates differ only slightly from the previous positions.

Require: Input data $X = \{\mathbf{x}_i\}_{i=1}^N$, kernel $k(\cdot, \cdot)$ and bandwidth h.

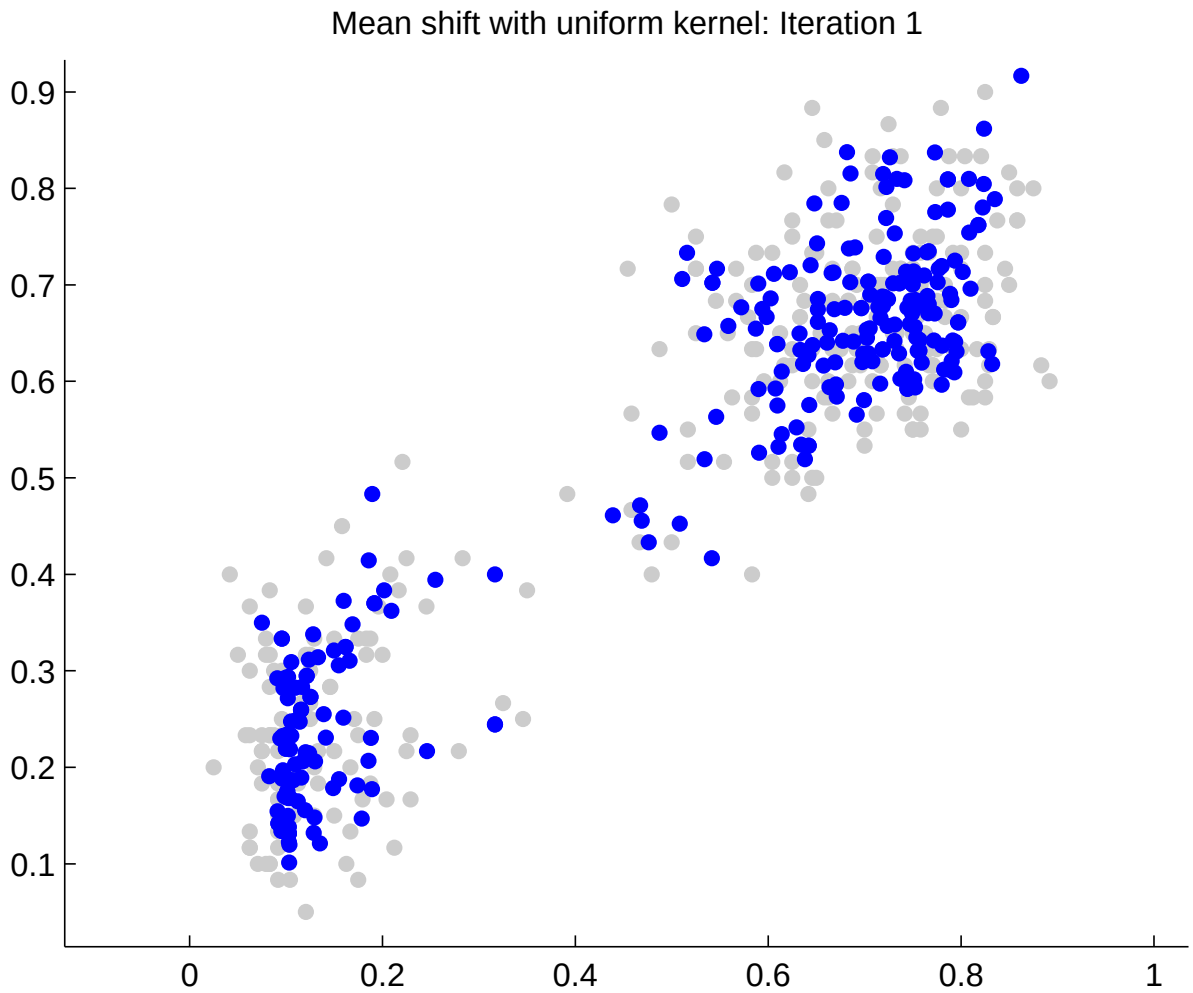
1: Initialise N cluster centres $\{\mathbf{y}_i\}_{i=1}^N$ with each $\mathbf{y}_i = \mathbf{x}_i$.
2: **while** Cluster centres have not converged **do**
3: **for** $i = 1, 2, \ldots, N$ **do**
4: Compute distance of $\mathbf{y}_i$ to all points in X.
5: Apply kernel function on $\mathbf{y}_i$ and all points in X.
6: Perform mean shift update on $\mathbf{y}_i$.
7: **end for**
8: **end while**

Mean shift algorithm (cont.)

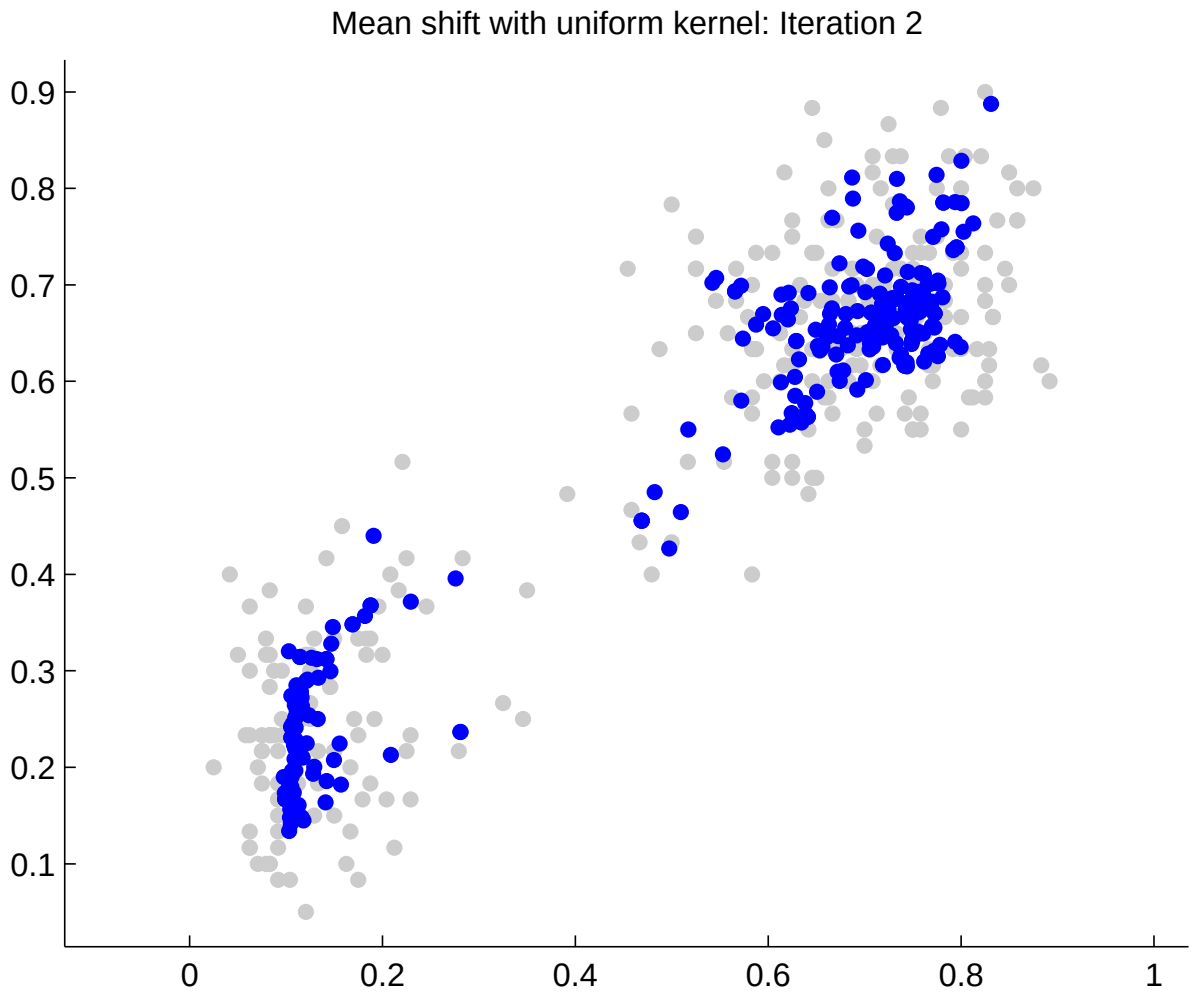
Example: Mean shift on the Old Faithful dataset with the uniform kernel and $h = 0.1$.



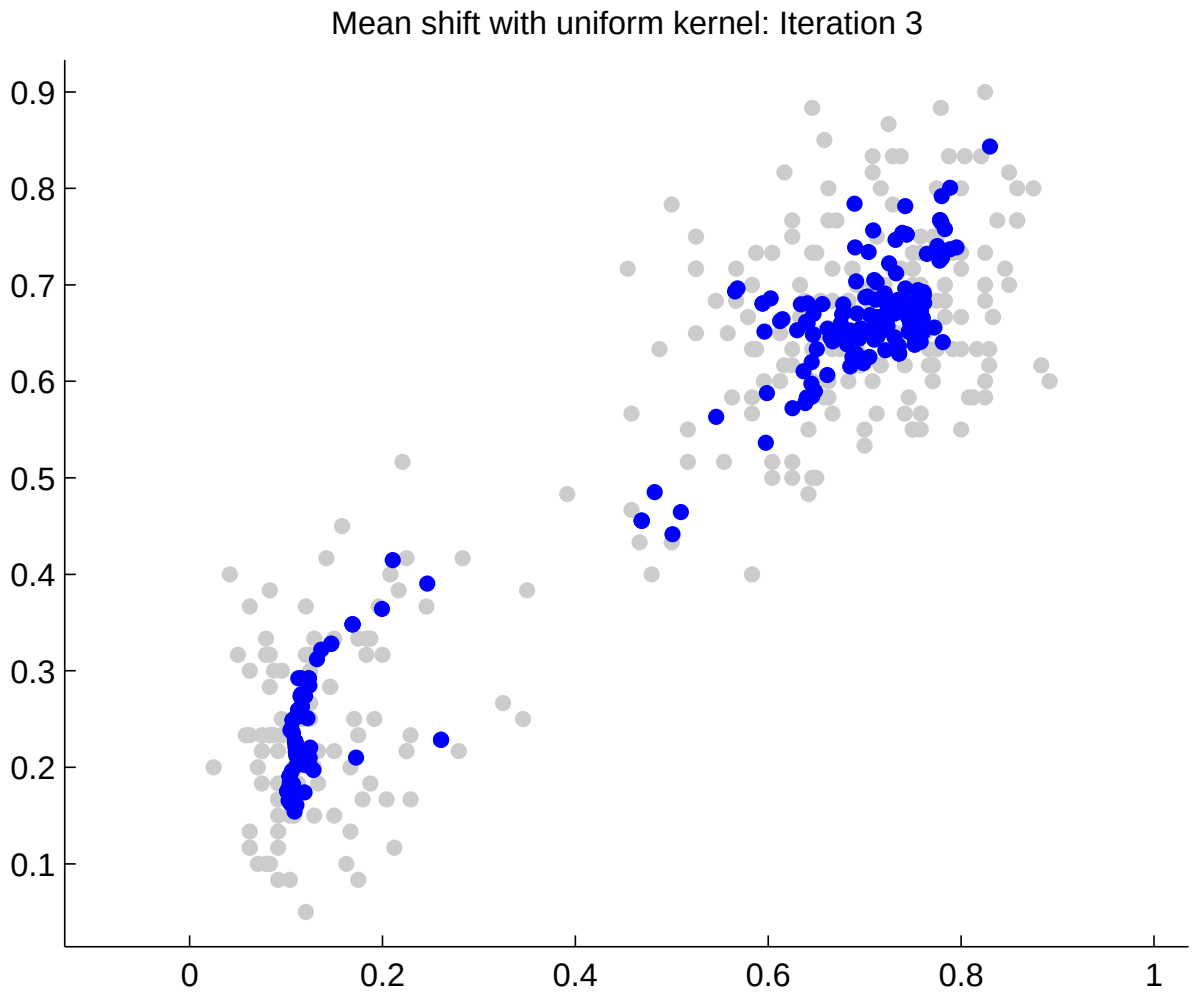
Mean shift algorithm (cont.)



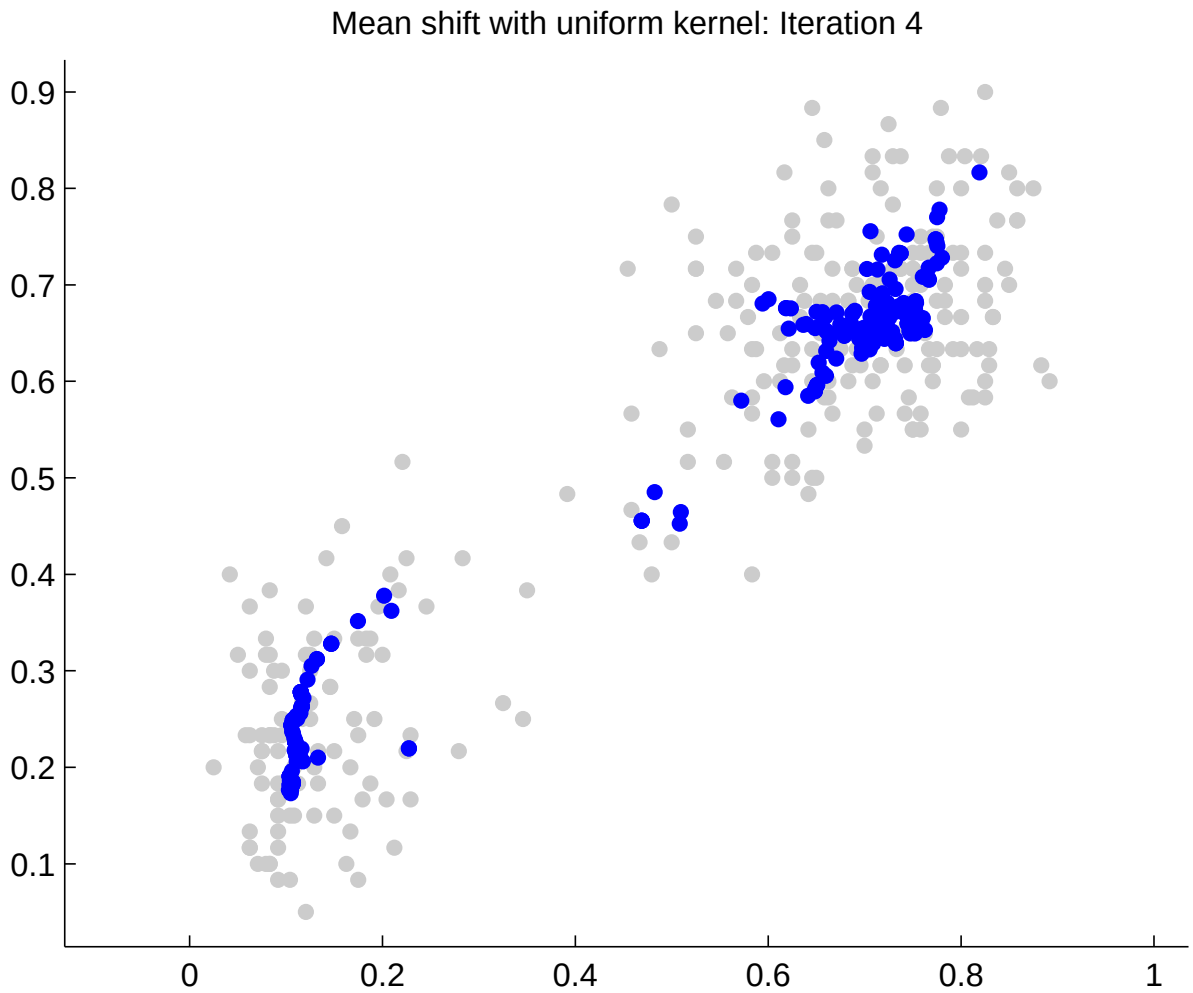
Mean shift algorithm (cont.)



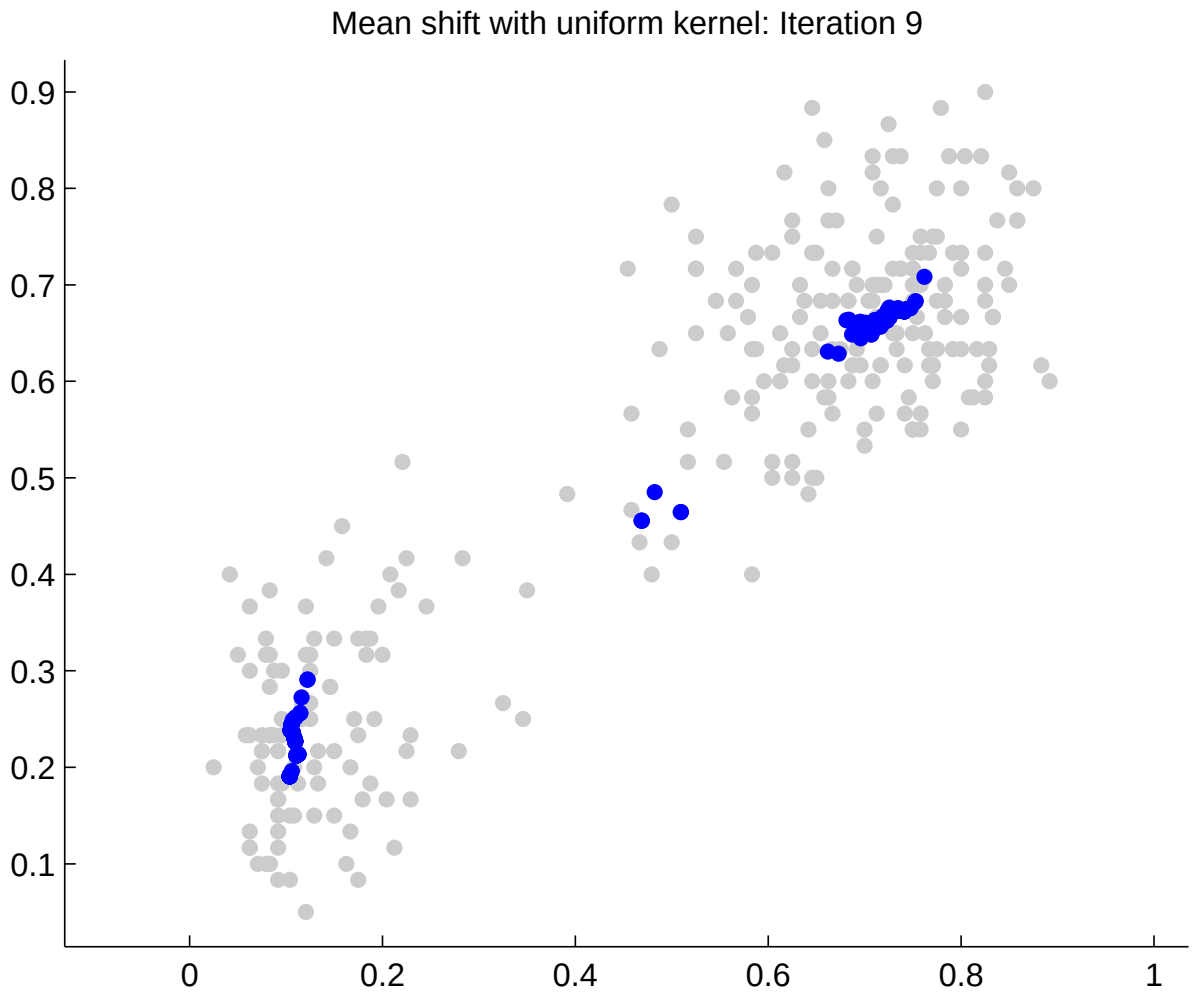
Mean shift algorithm (cont.)



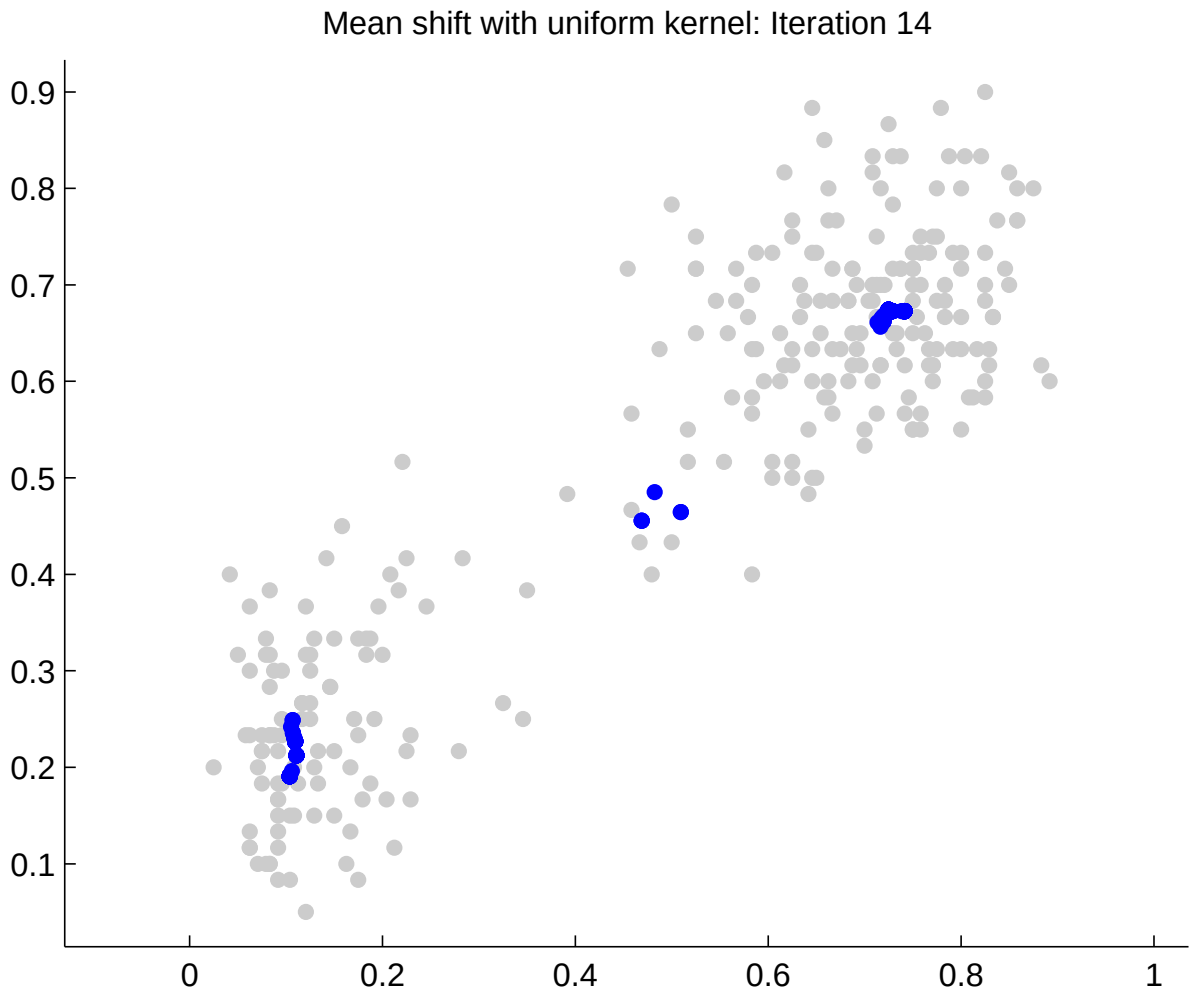
Mean shift algorithm (cont.)



Mean shift algorithm (cont.)



Mean shift algorithm (cont.)



Mean shift algorithm (cont.)

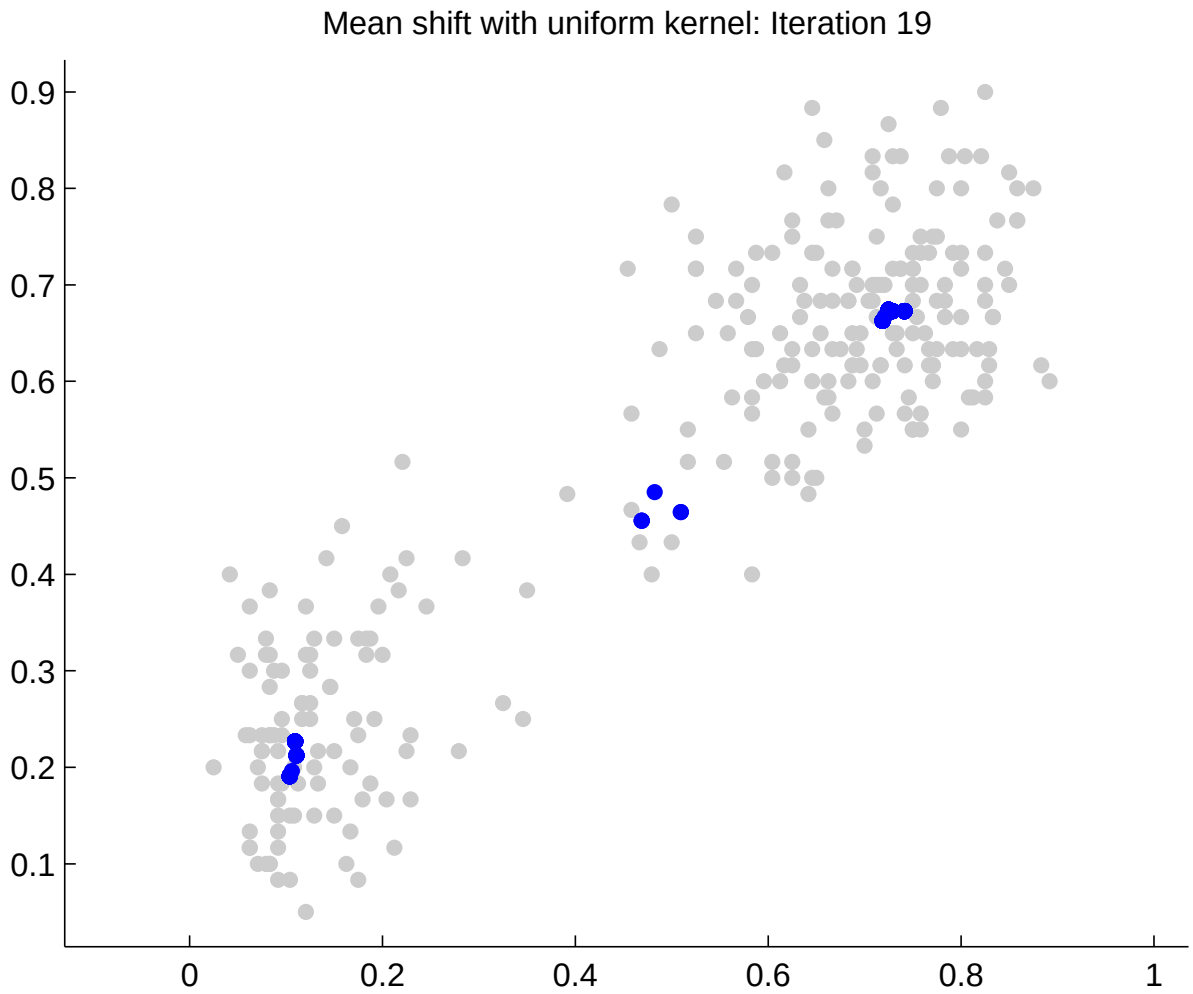


Image segmentation with mean shift