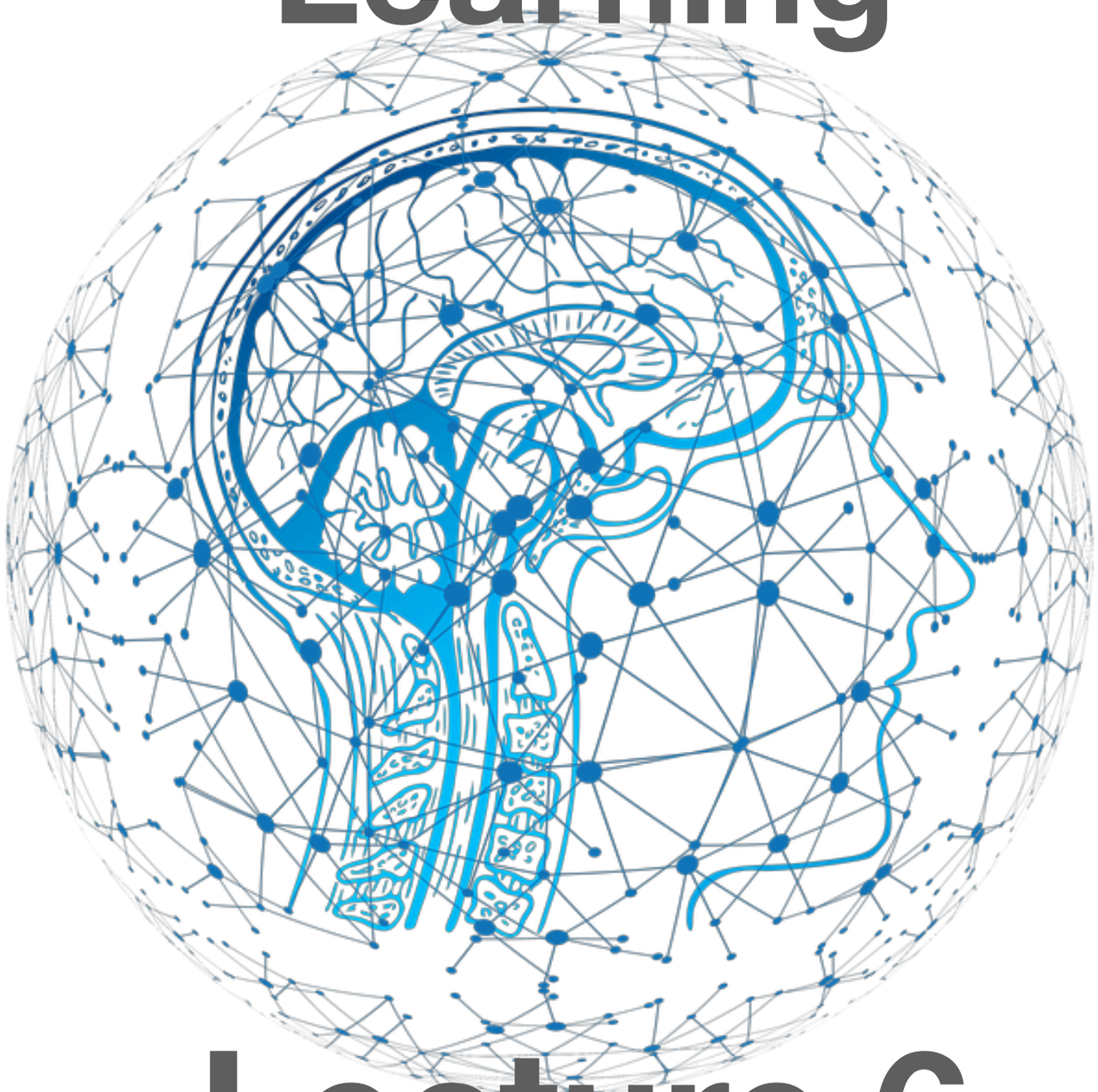
Course 395: Introduction to Machine Learning



Lecture 6

Dr Antoine Cully

Course 395:
Introduction to Machine Learning

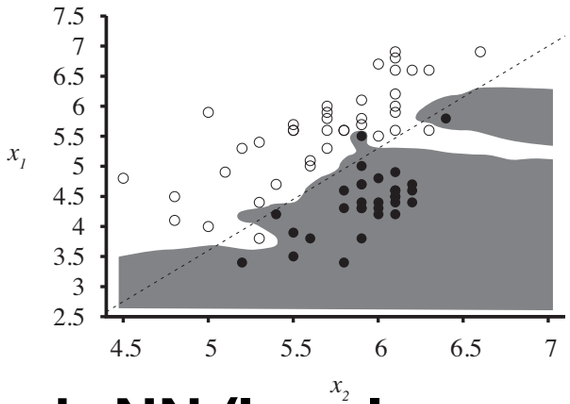
Lab sessions every week, starting next week.

- Week 2: Intro and Instance Based Learning (*A. Cully*)
- Week 3: Decision Trees & CBC Intro (*A. Cully*)
- Week 4: Evaluating Hypotheses (*A. Cully*)
- Week 5: Artificial Neural Networks I (*N. Cingillioglu*)
- Week 7: Artificial Neural Networks II (*N. Cingillioglu*)
- **Week 8: Unsupervised Learning and Density Estimation (*A. Cully*)**
- Week 9: Genetic Algorithms (*A. Cully*)

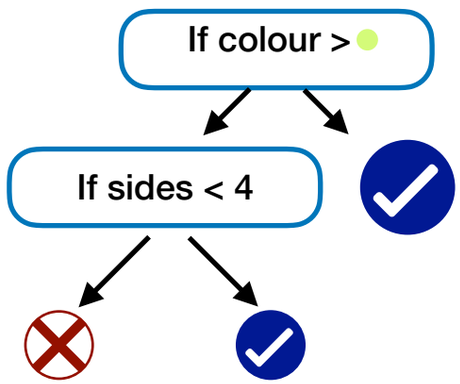
Lecture Overview?

- Unsupervised learning
- Clustering with k-means
- Density estimation with GMM
- K-means versus GMM

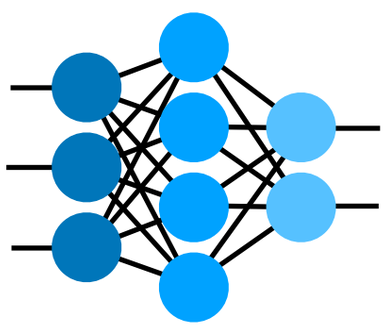
Recap so far…



k-NN (lazy learner)



Decision Trees (eager learner)



Neural Networks (eager learner)

Approach family:

Supervised Learning

Tasks:

- **Classification**
- **Regression**

Topic of today: **Unsupervised Learning**

Unsupervised learning

Unsupervised learning

- Usually done with **clustering**: trying to group data together in the feature space (there is no label space in unsupervised tasks).

- A cluster is a collection of data items which are “similar” between them, and “dissimilar” to data items in other clusters.

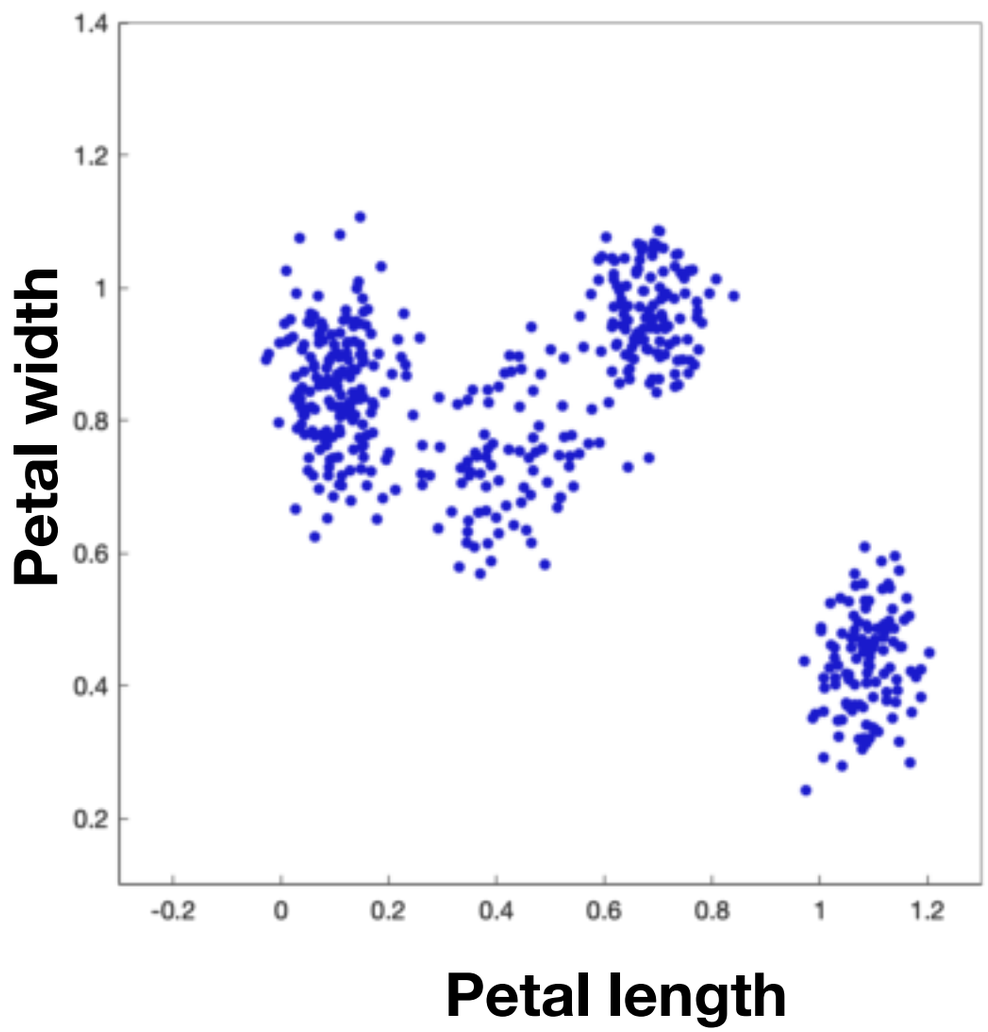
- A simple (and yet effective) approach is k-means.

Let’s take an example

- You have collected some data about several flowers that you have observed that look somewhat similar, but different at the same time.

- You want to know if they all belongs to the same specie or if there are actually several species.

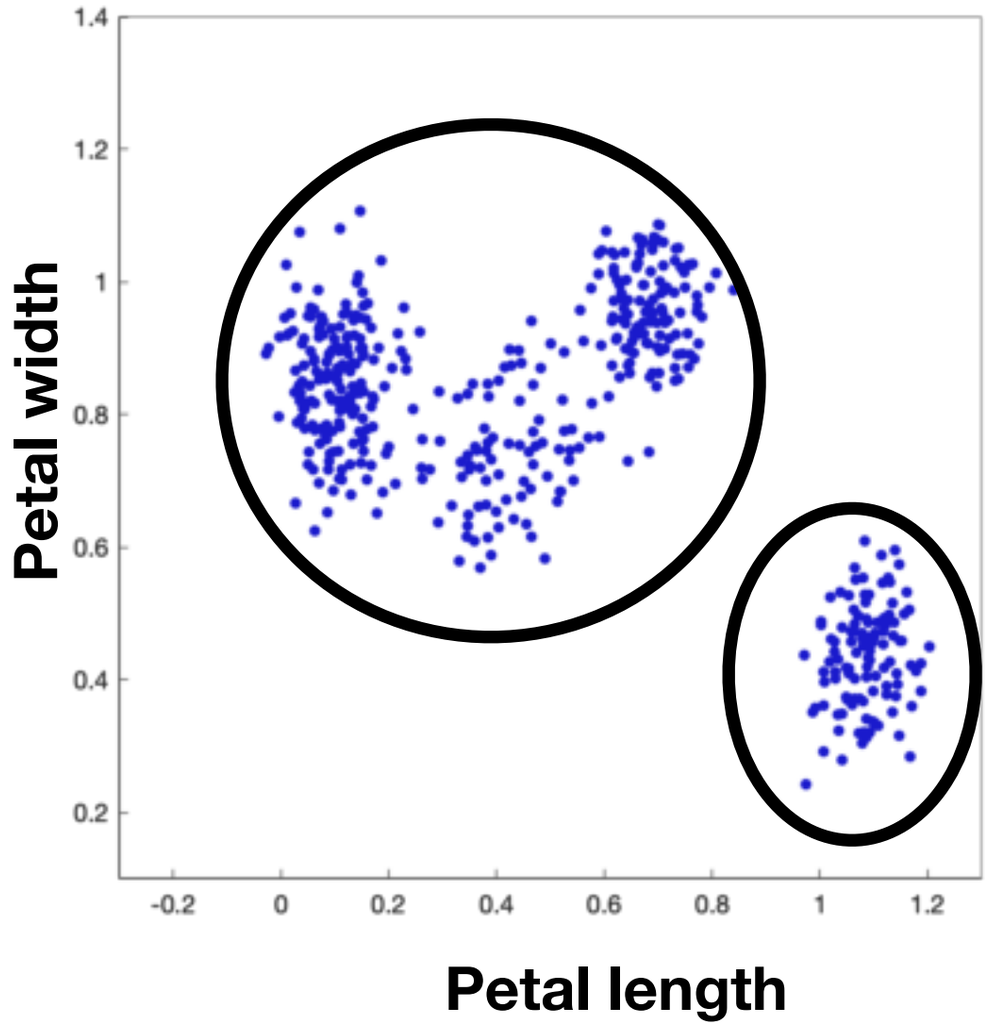
- The right label in this case does not exist (the name of each flower is not written under its petals). This is an unsupervised learning problem.



Objective

- The data shows a specific structure that might suggest that we observed several species

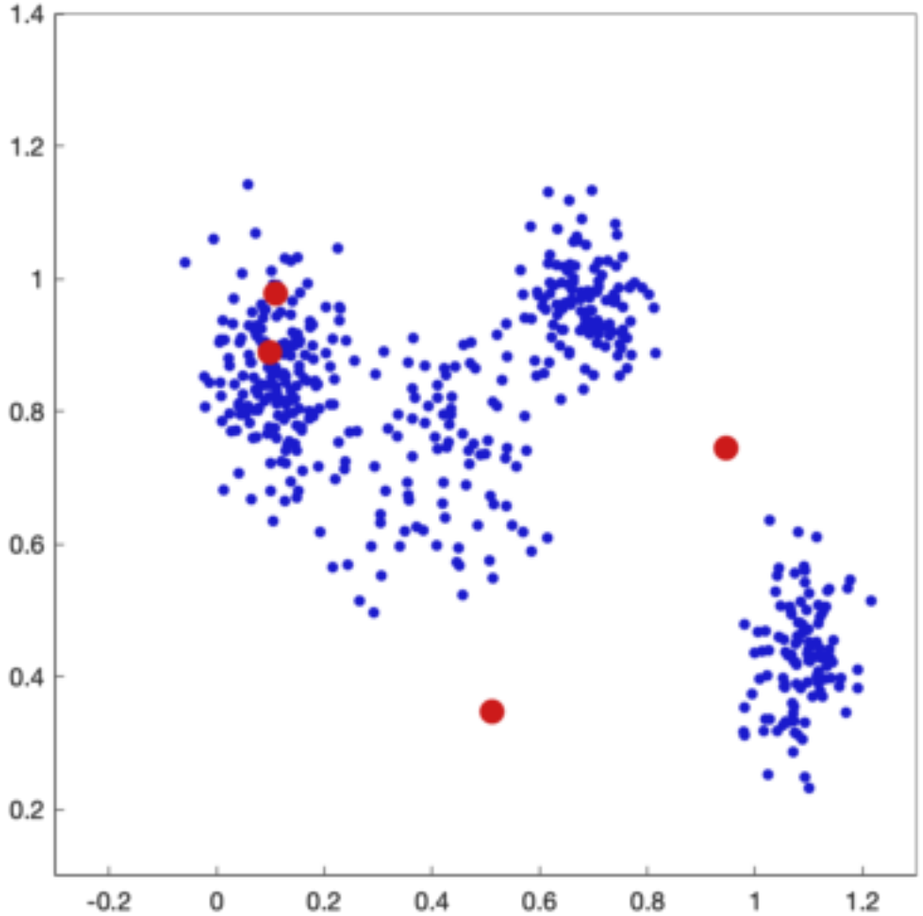
- **Can we automatically cluster the data points to form classes?**



k-means

Initialisation:

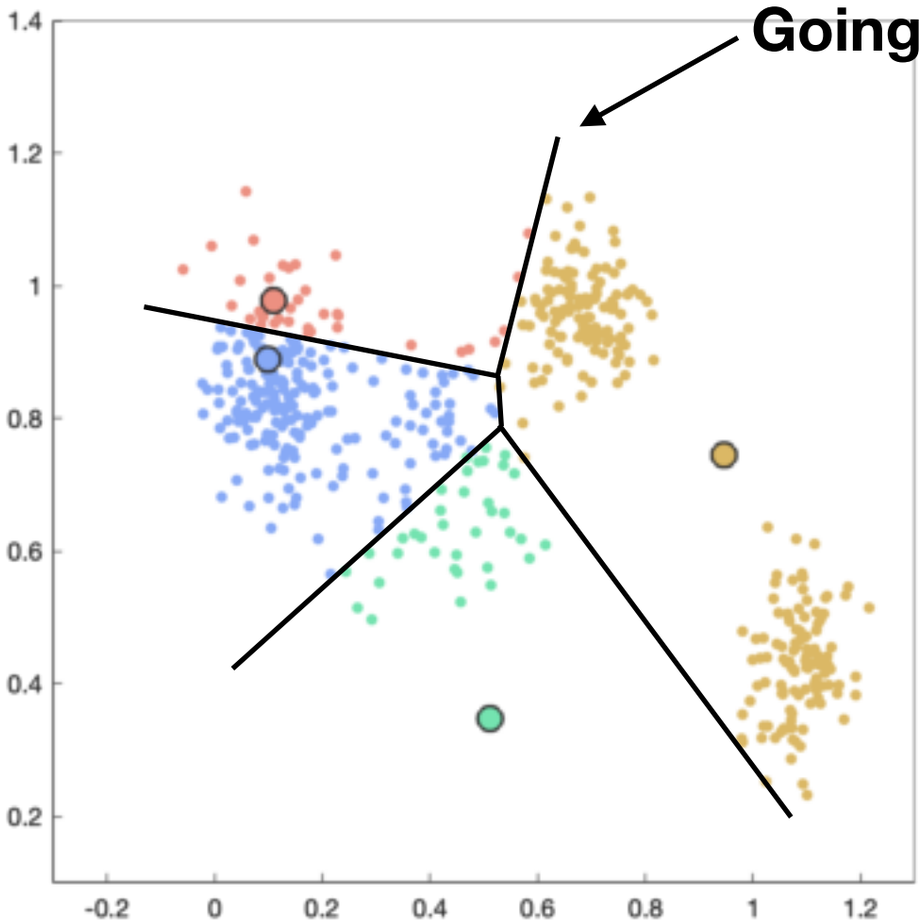
- 1) Select the number of cluster: **k**
- 2) Randomly place k centroids in you feature space:



k-means

Assignment:

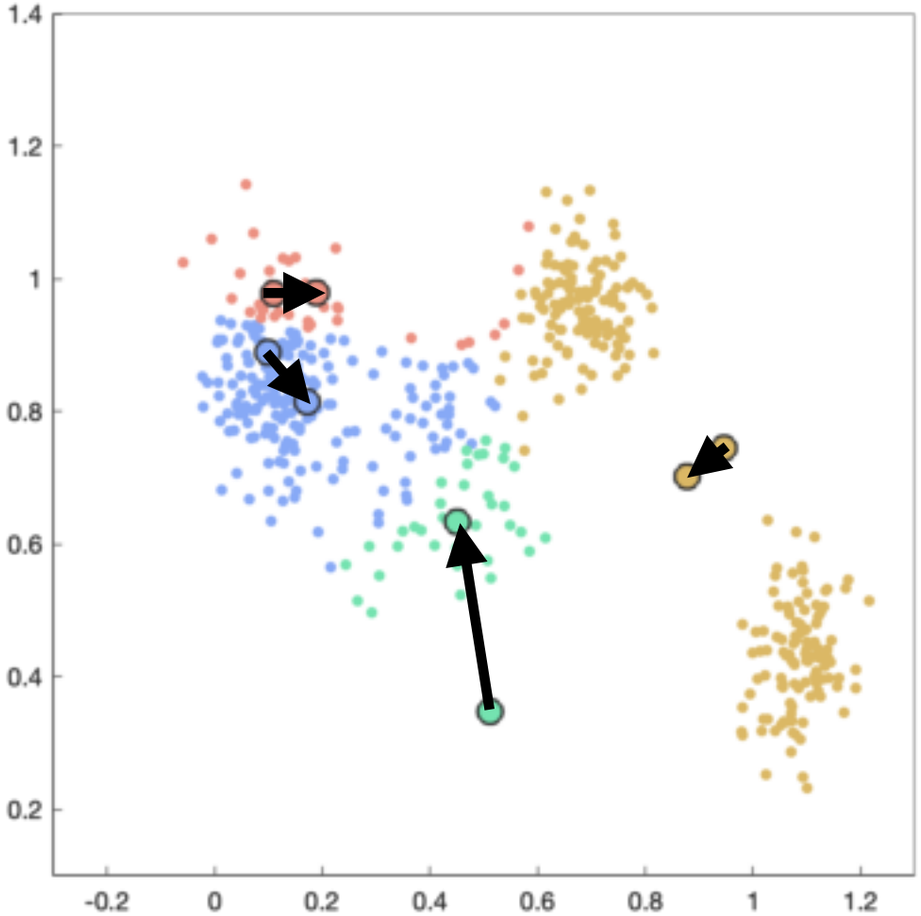
- Assign each datapoint to the nearest centroid.
  (for this you need to define a distance function, usually the euclidean distance, but other functions can be considered.)



k-means

Update:

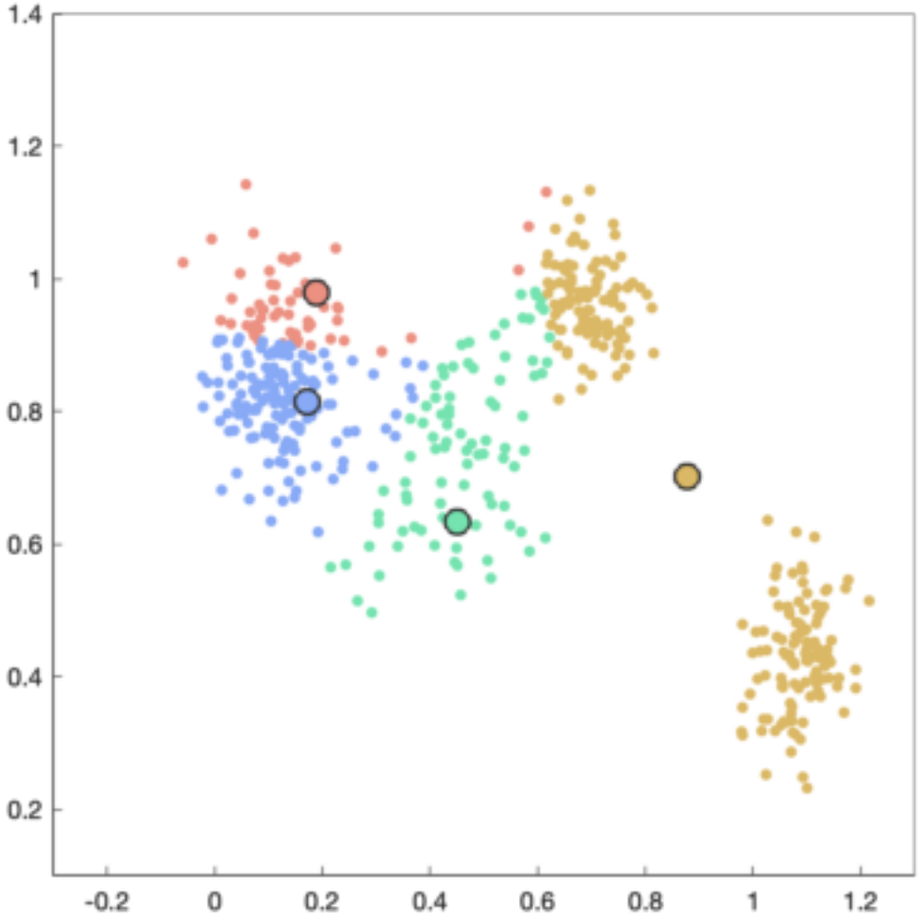
- Update the position of each centroid by computing the mean position of all the datapoints associated to the centroid.



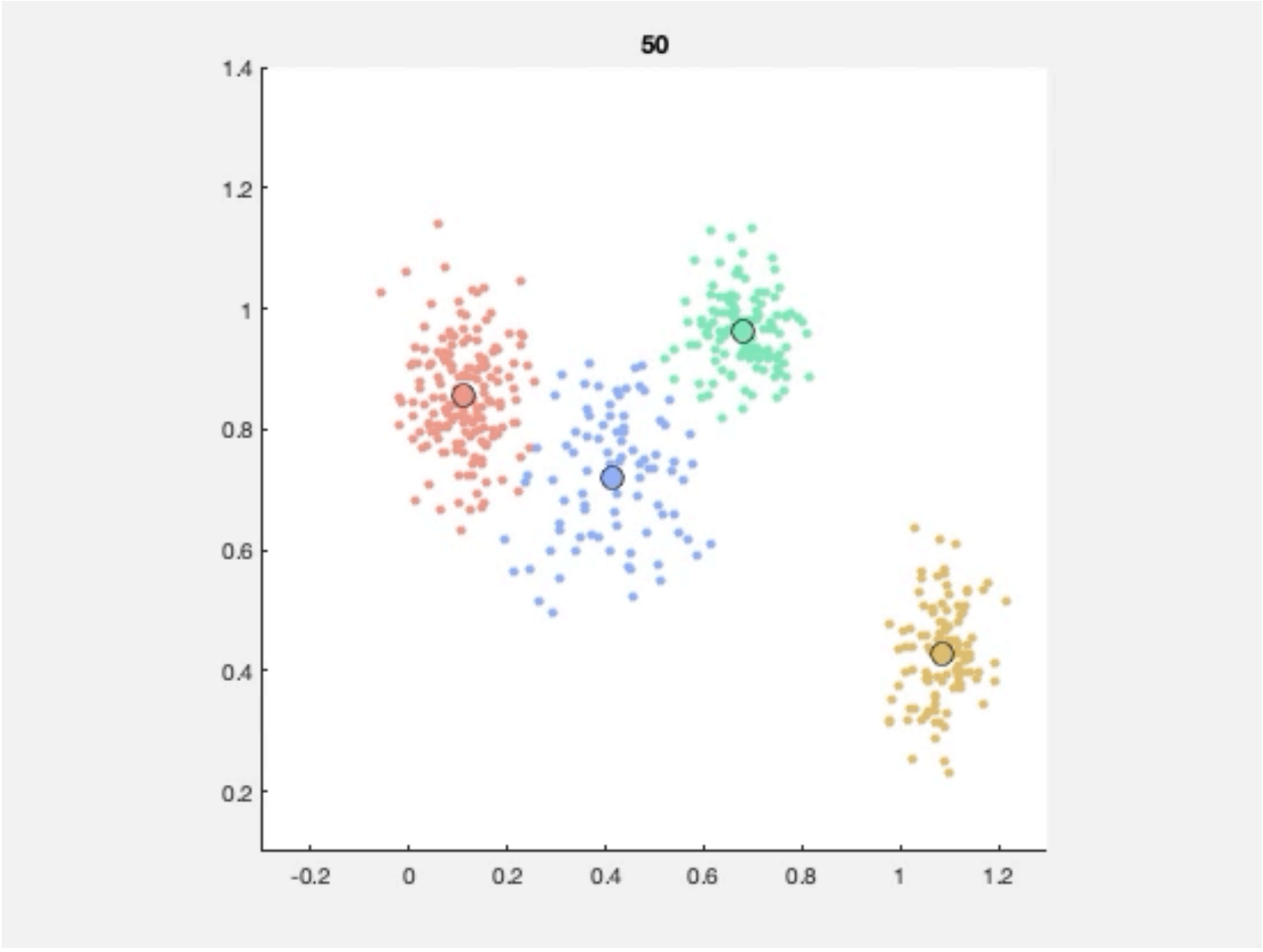
k-means

Repeat:

- The positions of the centroids have changed, so the assignments might not be valid anymore. So, repeat assignment and update until convergence.



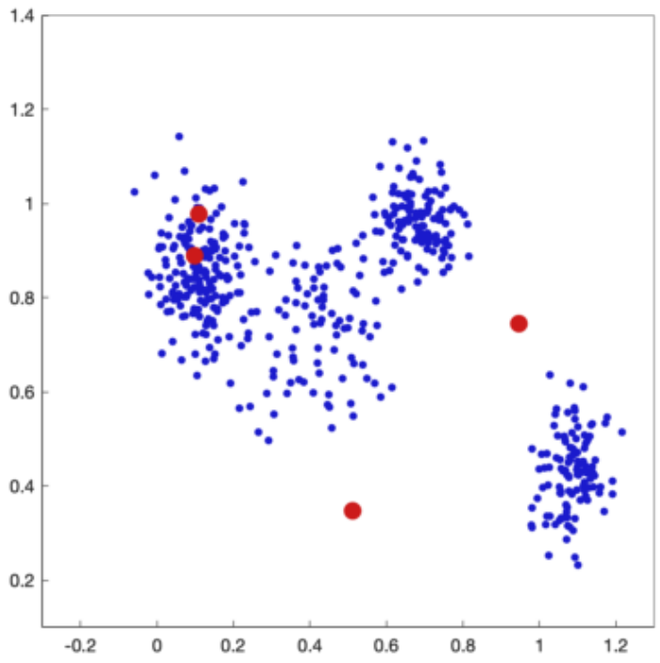
K-means in video



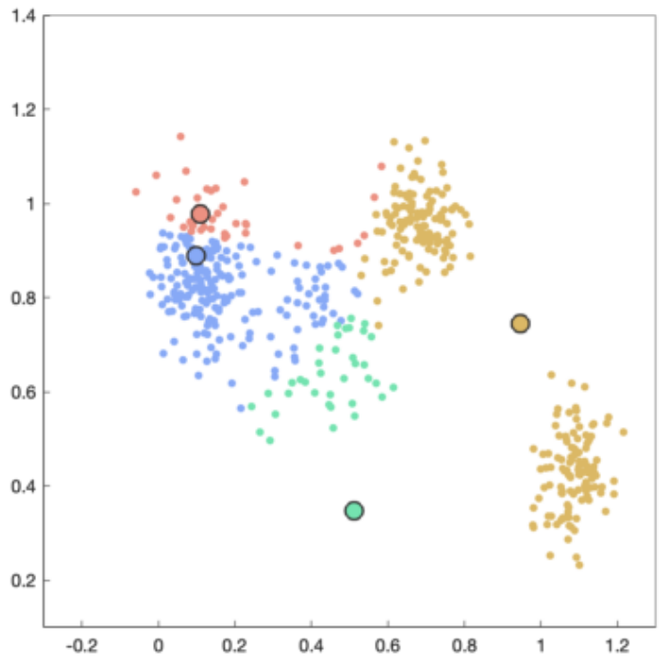
K-means algorithm

k-means:

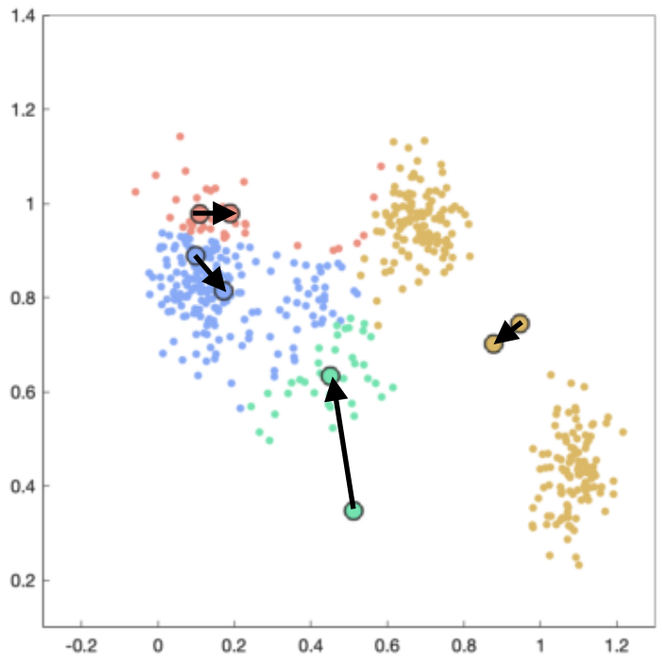
1) Initialisation
2) Assignment
3) Update centroids
4) If centroids moved return to 2)



Initialisation



Assignment

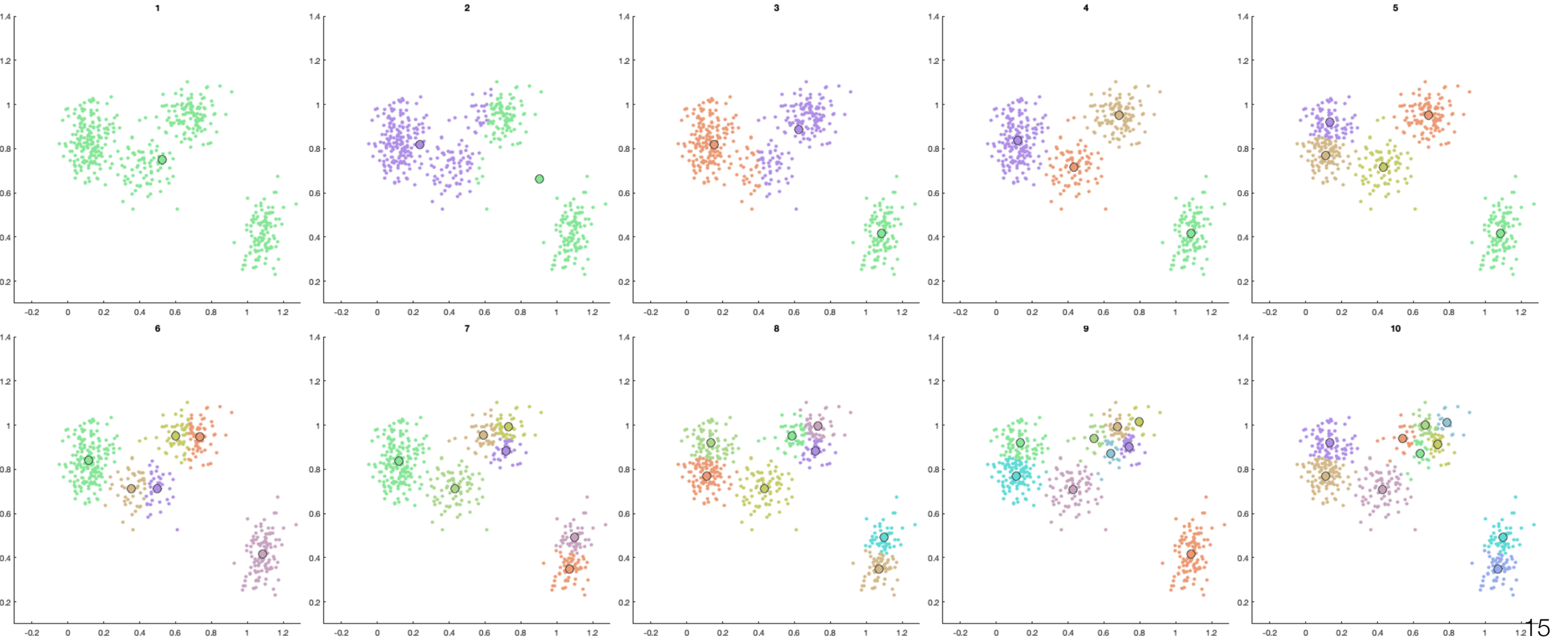


Update

Selection of k: elbow method

Several approaches exist to select k.
The main one is the "**elbow method**":

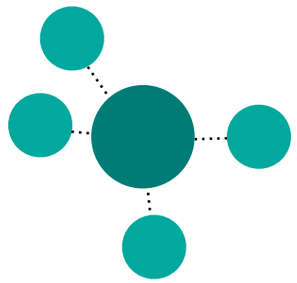
- Run k-means for all the possible values of k (e.g., from 1 to 10)



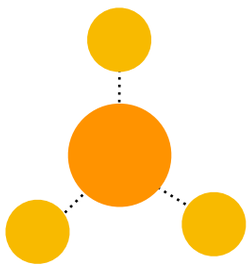
Selection of k:
The elbow method

Several approaches exist to select k.
The main one is the "**elbow method**":

- Run k-means for all the possible values of k (e.g., from 1 to 10)
- Compute a score for each of them: average of the mean distance to centroid

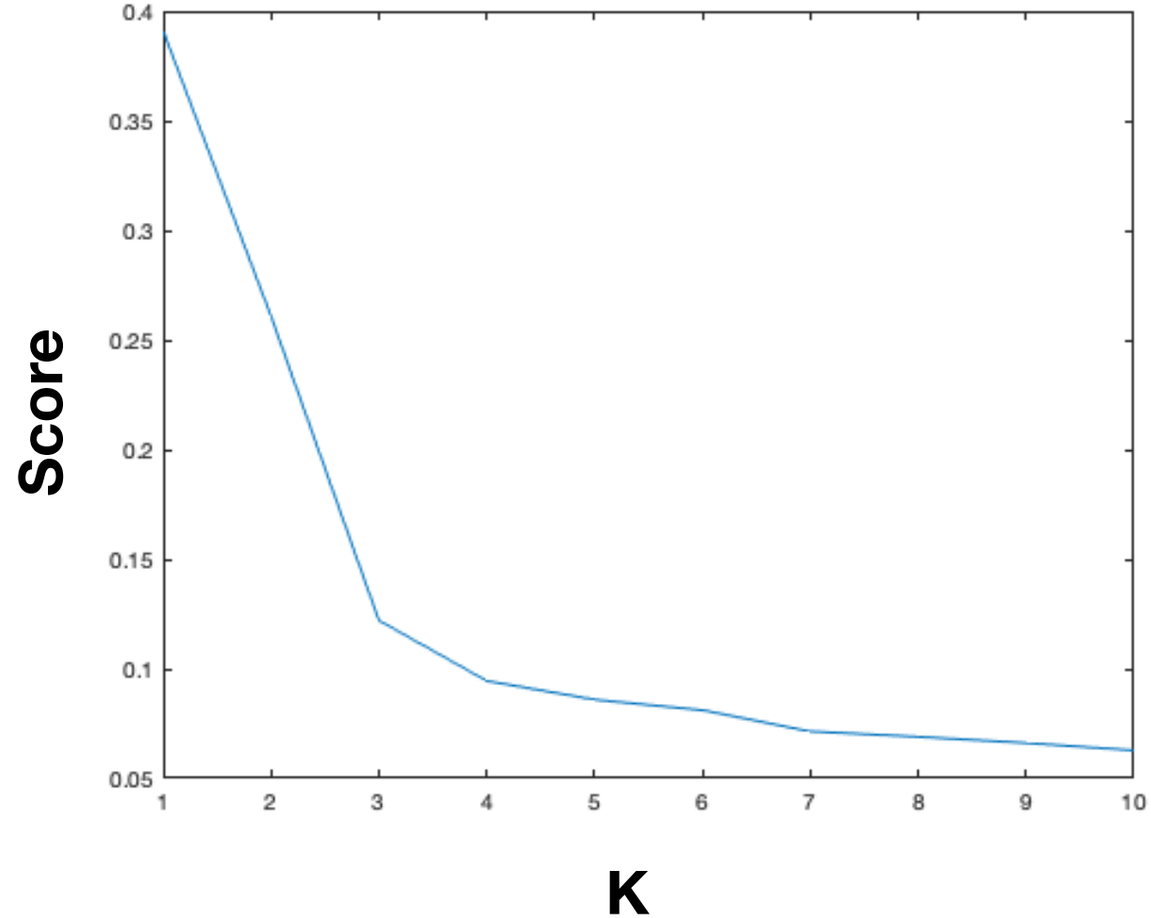


Mean distance_1



Mean_distance_2

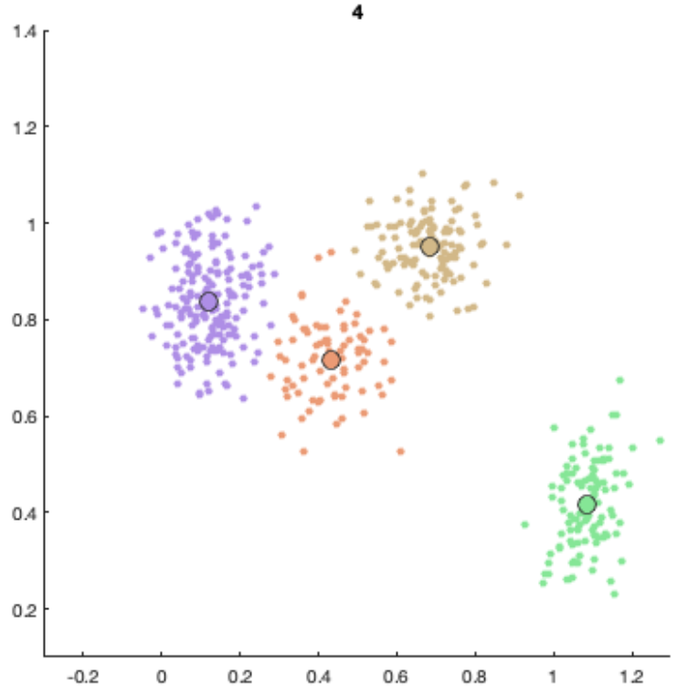
$$score = \frac{1}{K}\sum_{k=1}^{K} mean\_distance\_k$$



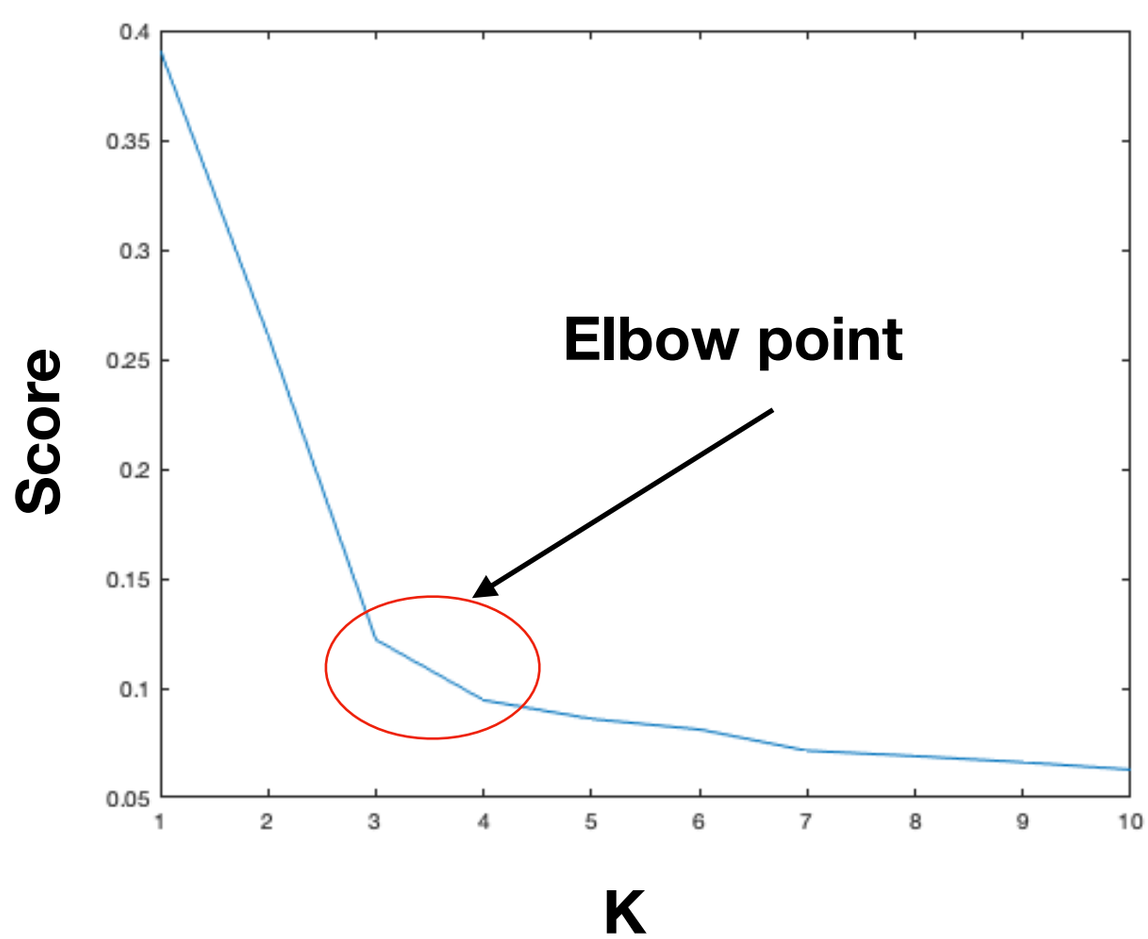
Selection of k:
The elbow method

Several approaches exist to select k.
The main one is the "**elbow method**":

- Run k-means for all the possible values of k (e.g., from 1 to 10)
- Compute a score for each of them: average of the mean distance to centroid
- Select k where the rate of decrease sharply shifts



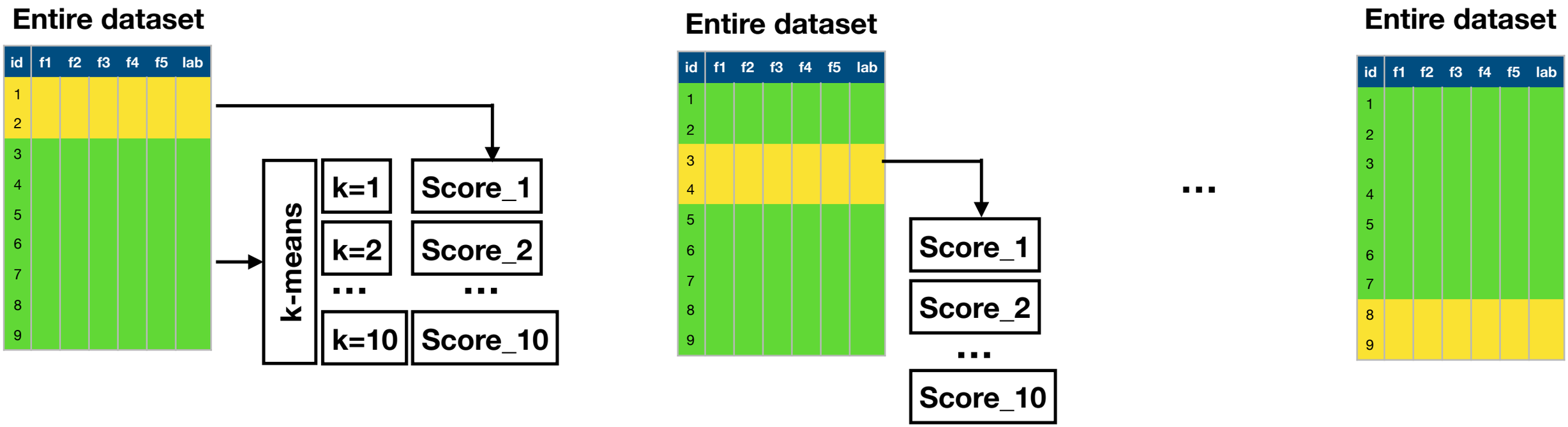
The score alone cannot be used, as it will always converges to 0 when k is equal to the number of datapoints.



Selection of k: Cross-Validation

Another approach is based on cross-validation.

The dataset is split in N folds, and N-1 folds are used to compute the centroids positions with k-means. Then, we compute the average score (same as before) on the validation datasets. We do this for various values of k and we pick the best configuration.



Select k such that further increase in number of clusters leads to only a small improvement in the average score_k.

Strengths of k-means

- **Simple**: easy to understand and to implement
- **Efficient**: Time complexity: O(tkn), where n is the number of data points, k is the number of clusters, and t is the number of iterations.Since both k and t are small (compared to n). k-means is considered a linear algorithm.
- **Popular**: K-means is the most popular clustering algorithm.
- Note: it only founds a local optimum (according to the score defined previously). The global optimum is hard to find due to complexity. Usually, we run several times k-means with random initialisation and we pick the best.

Weaknesses of k-means

- **We have to define k**

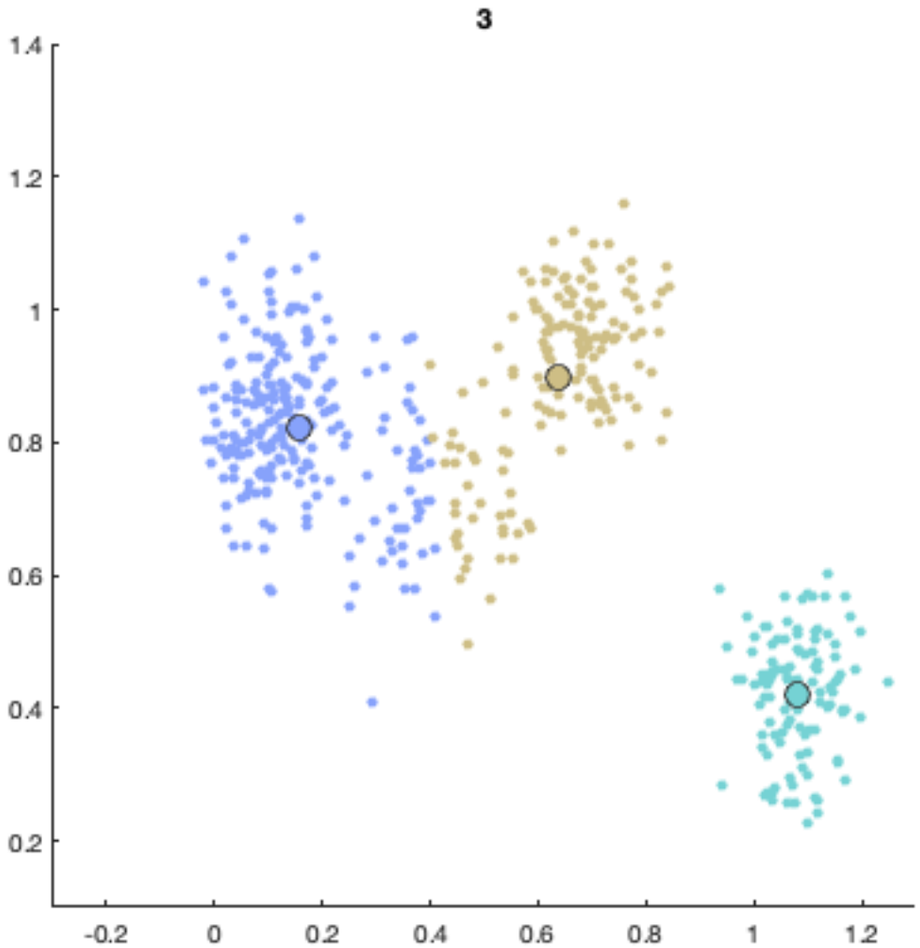
- The algorithm is only applicable if the mean is defined.
  A variant exist for categorical data: k-mode, the centroids represent the most frequent values.

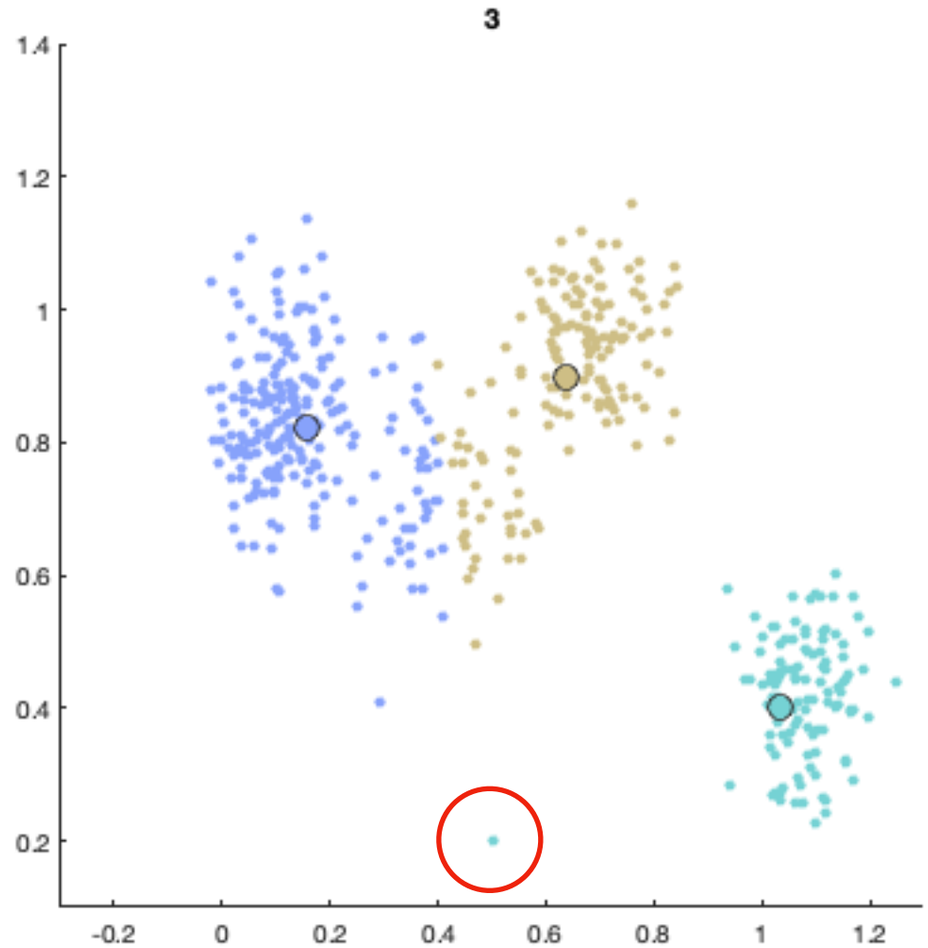
- Sensitive to initialisation (initial position of the centroids)

- The algorithm is sensitive to outliers (data points that are very far away from other data points).

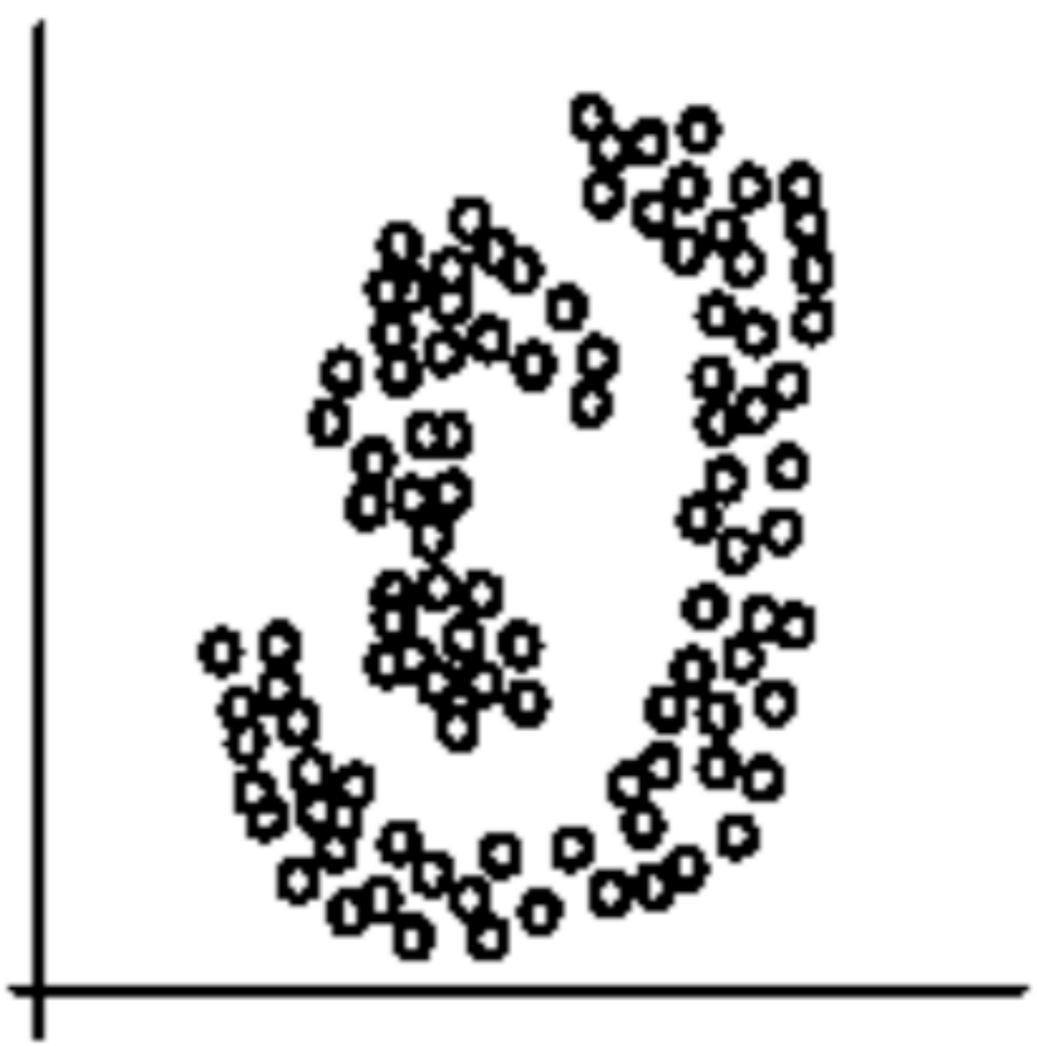
- The k-means algorithm is not suitable for discovering clusters that are not hyper-ellipsoids (or hyper-spheres).

Outliers

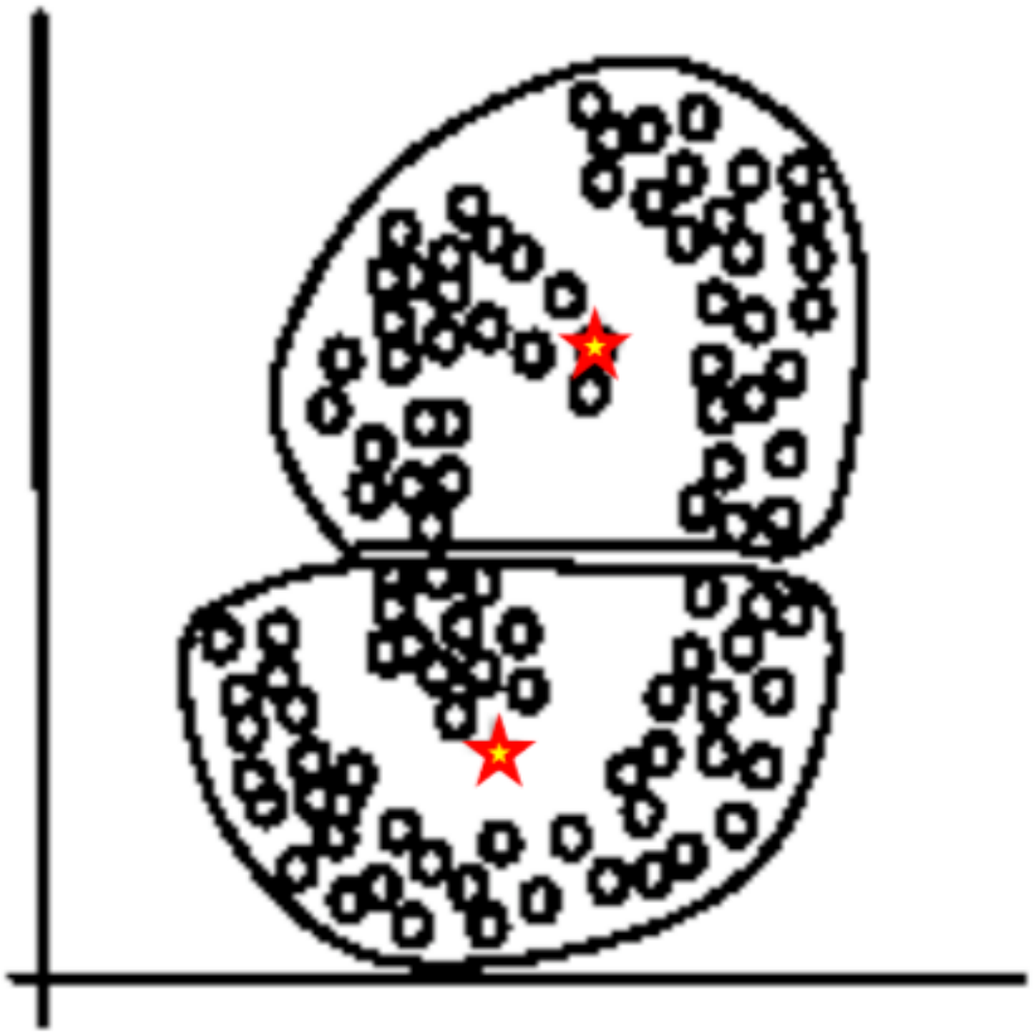

3
1.4
1.2
1
0.8
0.6
0.4
0.2
-0.2
0
0.2
0.4
0.6
0.8
1
1.2


3
1.4
1.2
1
0.8
0.6
0.4
0.2
-0.2
0
0.2
0.4
0.6
0.8
1
1.2

Hyper-ellipsoids



Natural Clusters



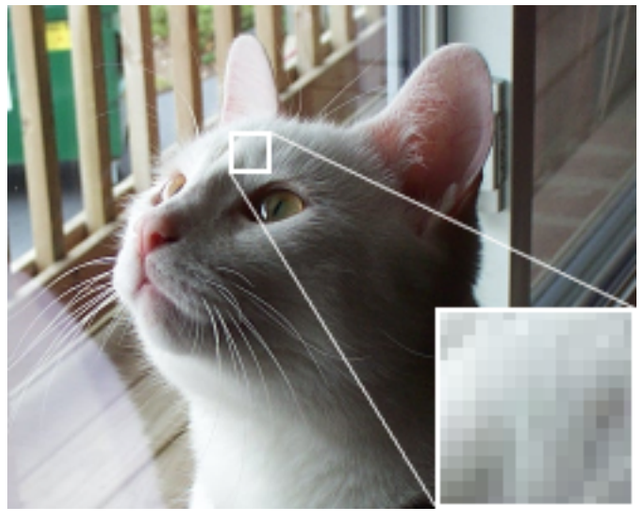
K-means Clusters

From 9.54 MIT

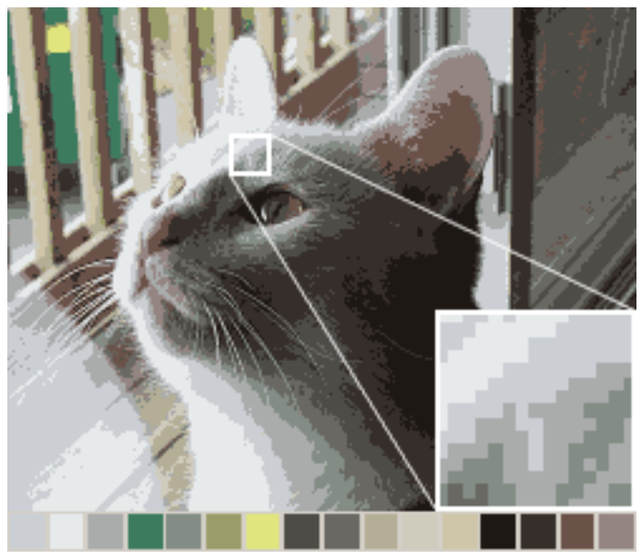
Applications:

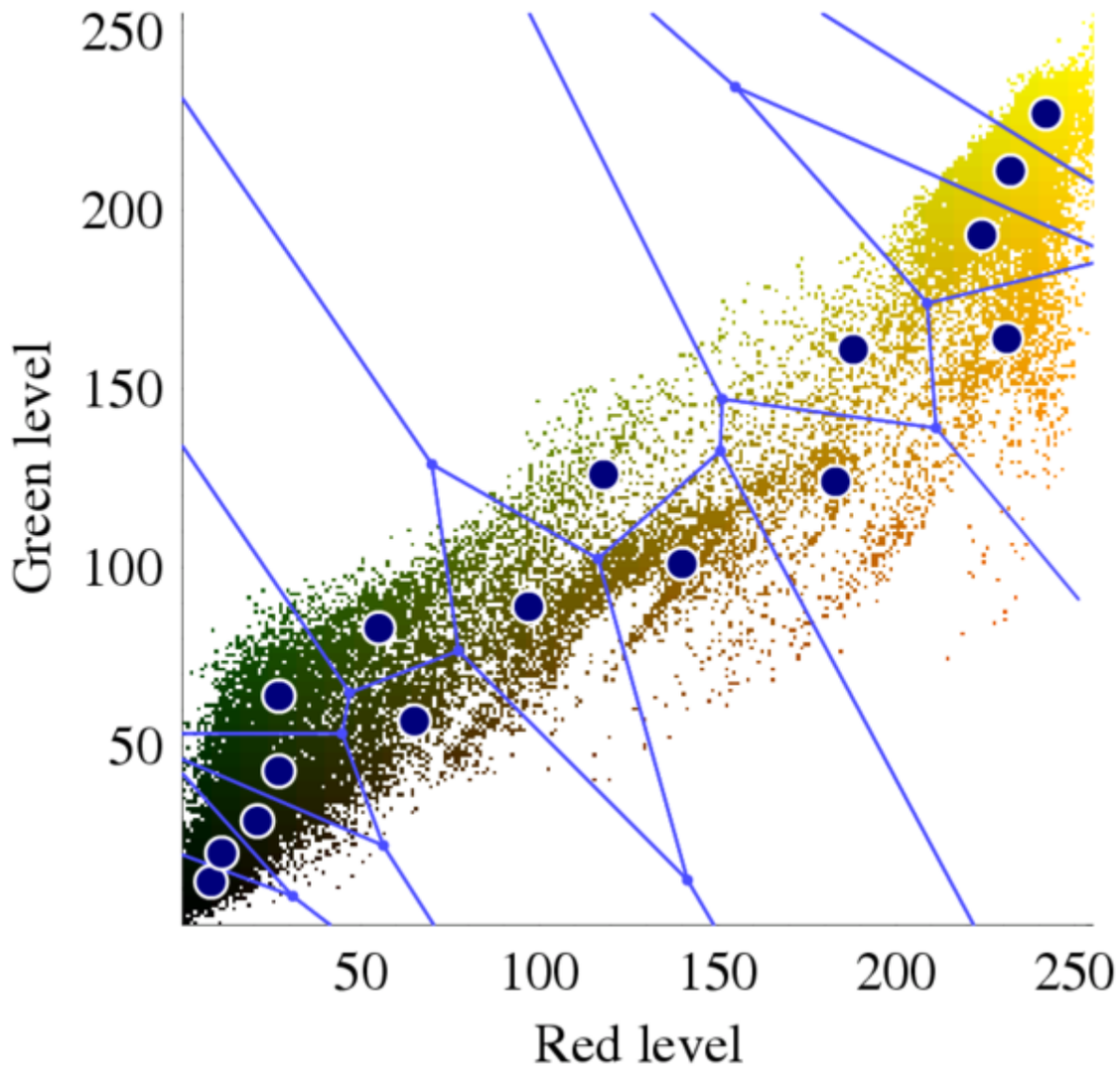
Vector quantisation

24bits color encoding



16 different colors (4 bits)

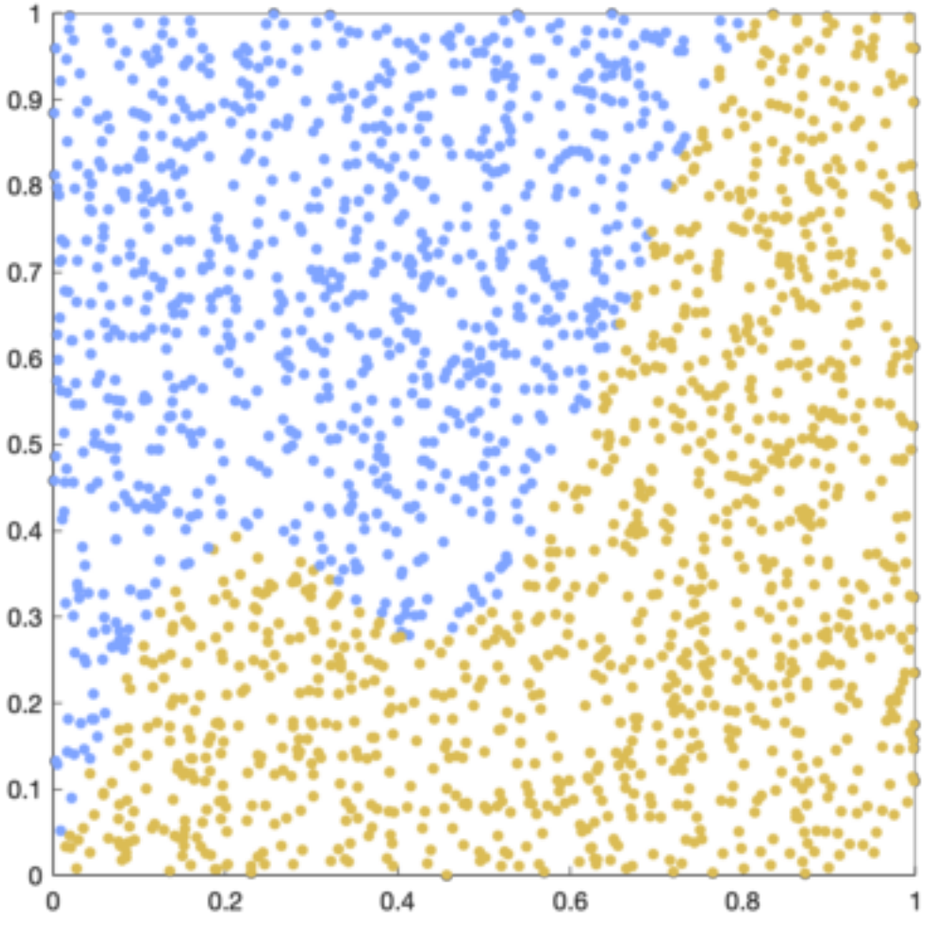




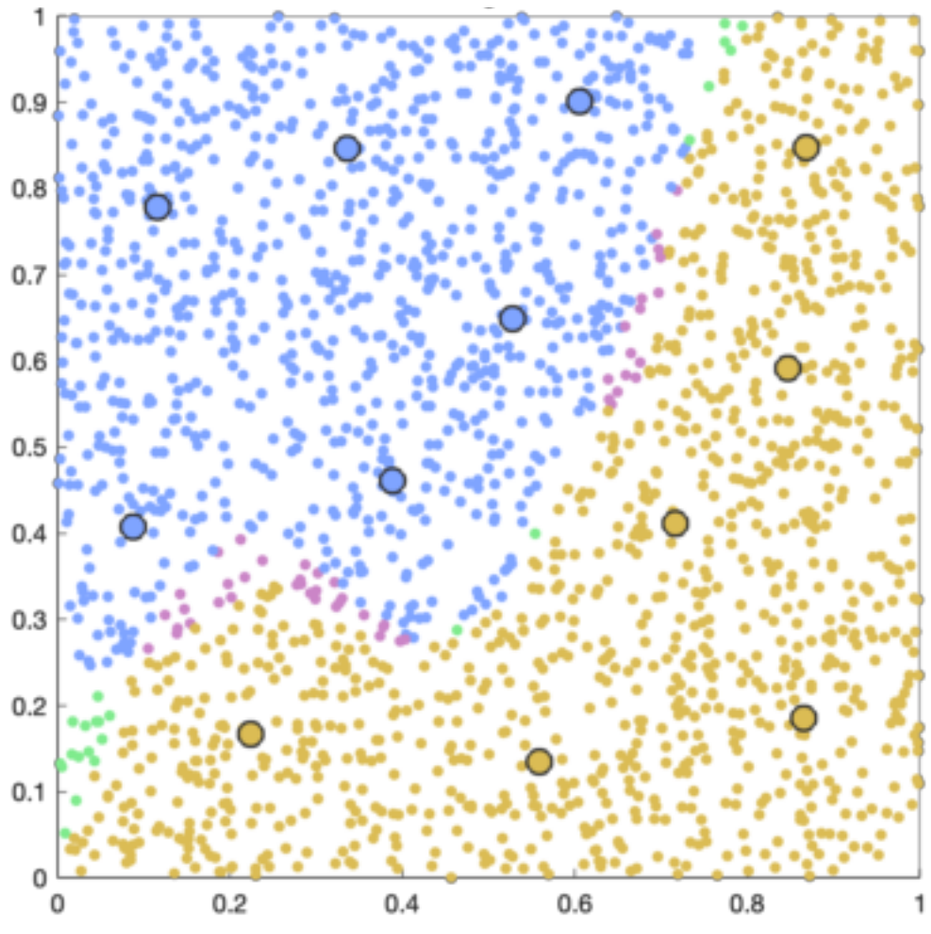
From Jamelan and Dcoetzee

Applications: Feature learning

Dataset: 2000 data points



Feature set: 12 features



Note: this is a naive application for feature learning, more advanced approaches exist.

Density Estimation

Motivations

Anomaly detection or novelty detection.

- Example: You want to detect the new trends on Youtube. You can use density estimation to create a model of the distribution of views over the different types of content.

- Thanks to this model, you will be able to detect if a type of content suddenly starts to accumulate more views than usual (novelty detection).

Motivations

- One of the main applications of density estimation is anomaly detection or novelty detection.

- **Let's take an another example:** monitoring jet engines.

- It is important to detect as soon as possible anomalies, to repaire/replace the engines when required.

- In particular, we can measure every minute, the current speed of the engine and its temperature.

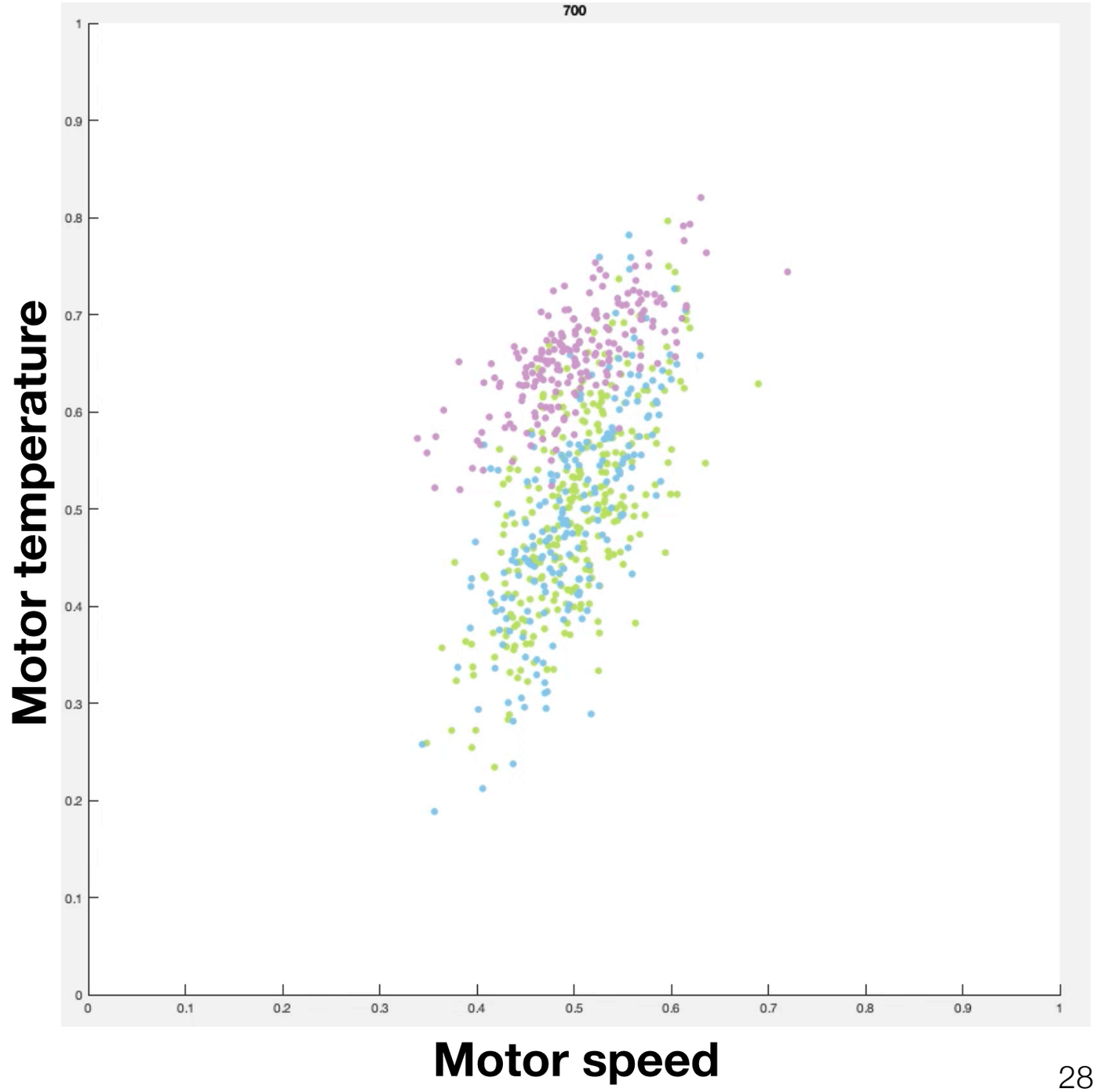
Motivations

- **Let's take an example:** monitoring jet engines.

- **Green points** are collected in factories (quality control)

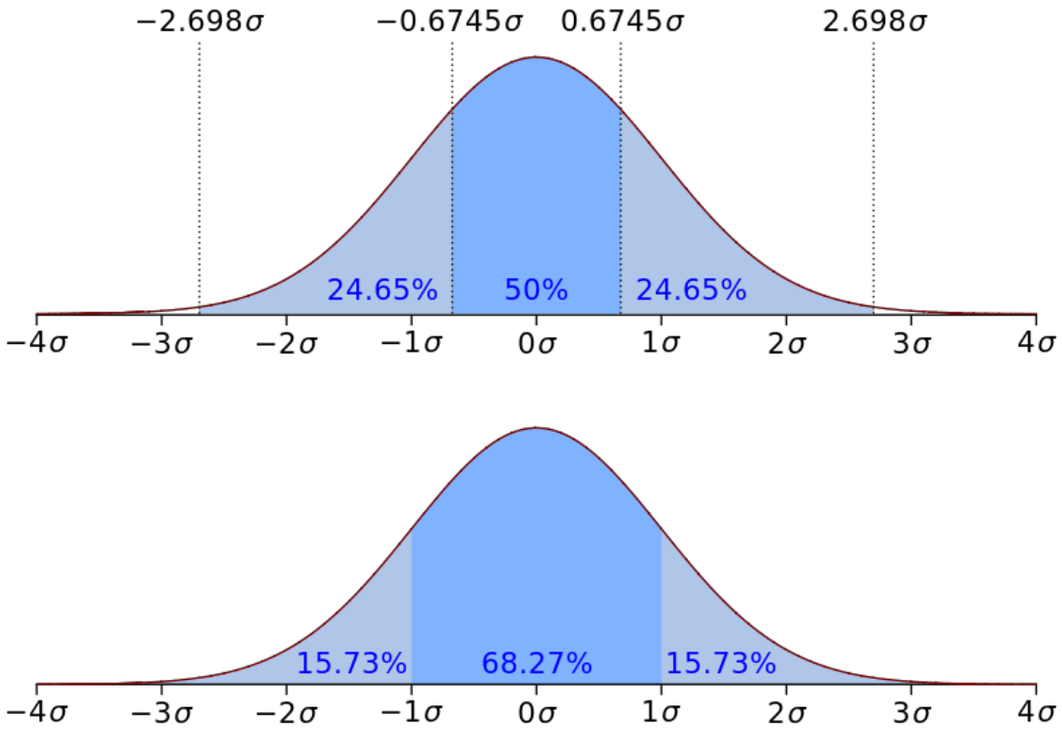
- **Blue points** are collected during flights on recent planes

- **Magenta points** are collected on a suspicious plane



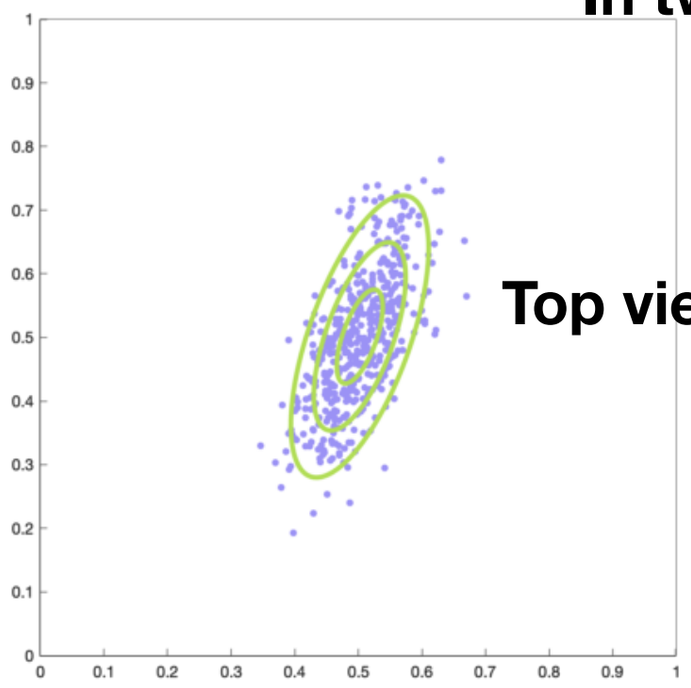
Probability Density Functions (PDF)

In one dimension

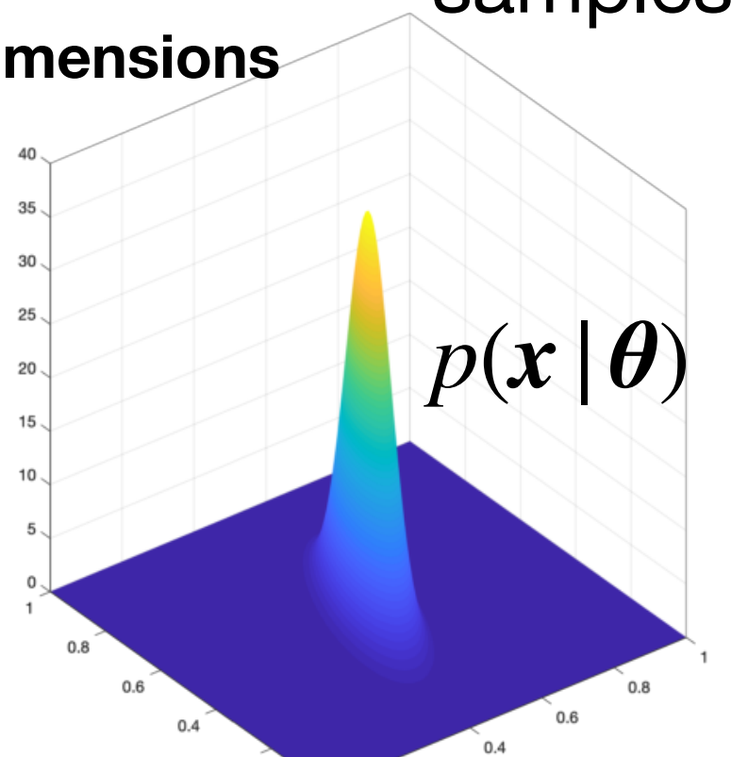


- In a few words: a PDF models the probability of a sample to be generated in a specific **area**.
- It is very **likely** or usual to observe samples where the **PDF is high.**
- Conversely, it is **rare** to observe samples where the **PDF is low.**

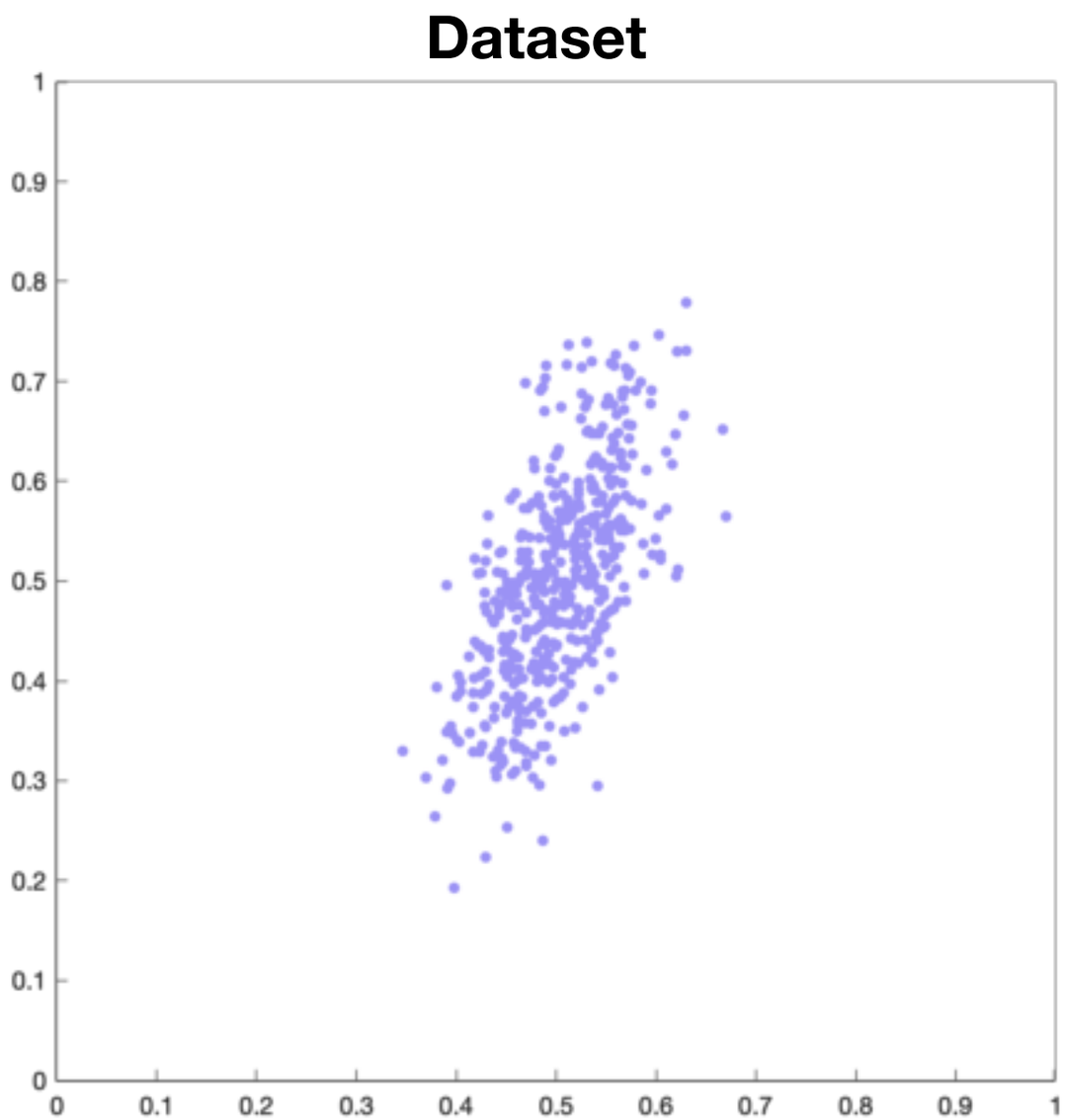
In two dimensions



Top view



Simple distribution



- We can see that some regions are denser than others.
- We can use a simple probability distribution to model this dataset: **Gaussian distribution.**
- The parameters of the Gaussian need to be fitted to the dataset.

Univariate:

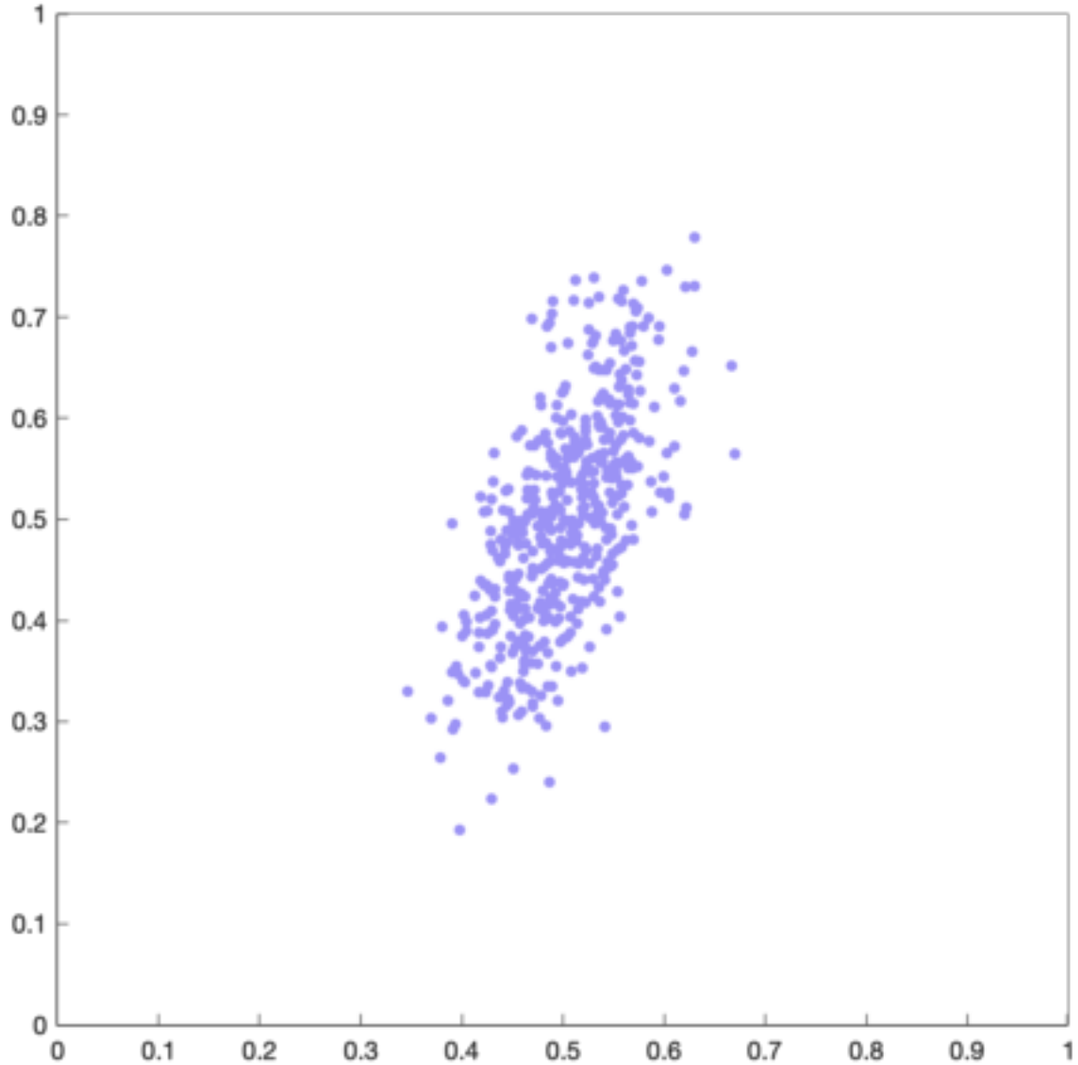
$$\mathcal{N}(x \mid \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} * e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Multivariate

$$\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^D \mid \boldsymbol{\Sigma} \mid}} * e^{-\frac{1}{2}(\mathbf{X}-\boldsymbol{\mu})'\boldsymbol{\Sigma}^{-1}(\mathbf{X}-\boldsymbol{\mu})}$$

D = number of dimensions

Finding the parameters



The Gaussian distribution has 2 parameters:

- The mean μ
- The (co)variance σ^2

Univariate:

$$\mu = \frac{1}{N}\sum_{n=0}^{N} x_n$$

$$\sigma^2 = \frac{1}{N}\sum_{n=0}^{N} (x_n - \mu)^2$$

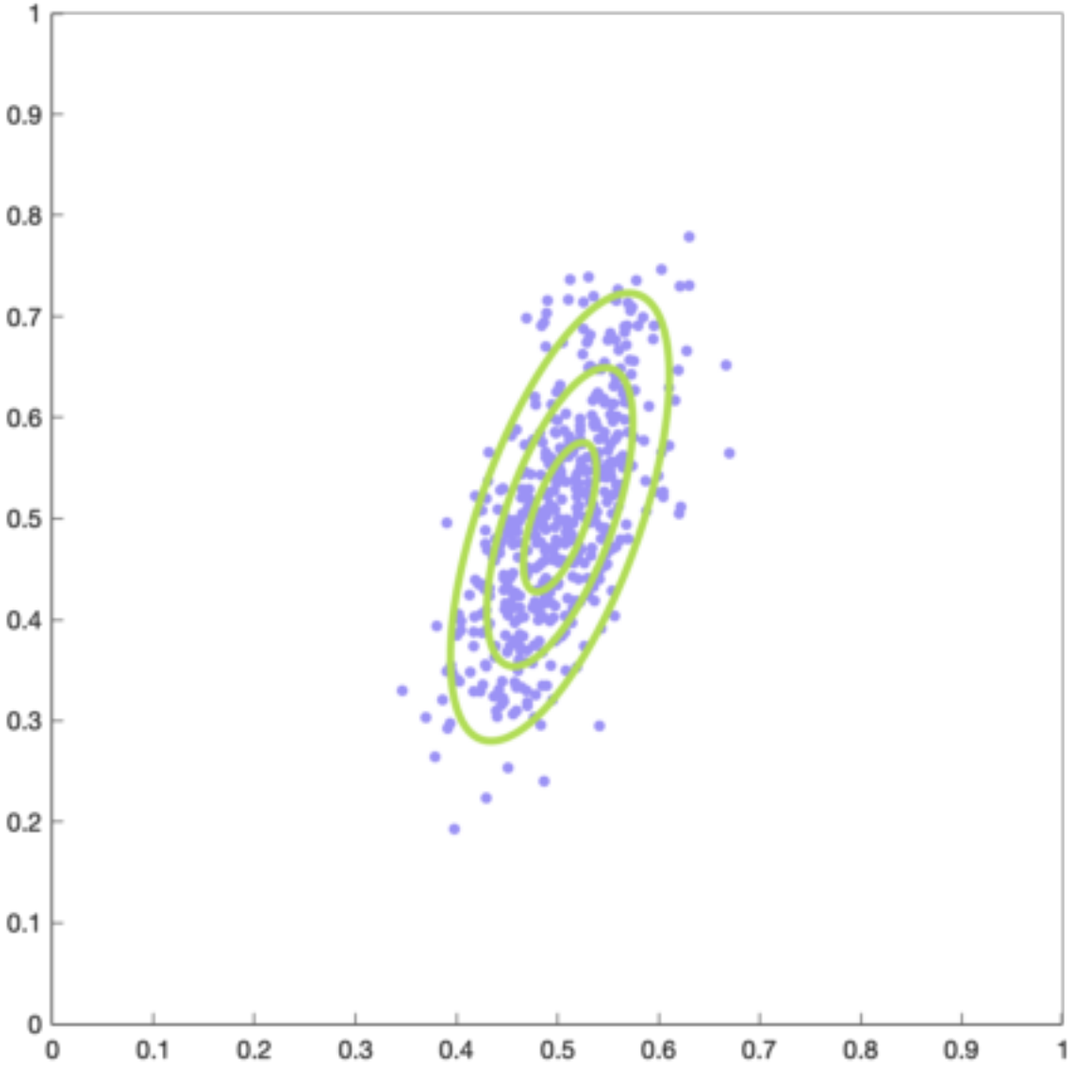
Multivariate

$$\boldsymbol{\mu} = \frac{1}{N}\sum_{n=0}^{N} \boldsymbol{x}_n$$

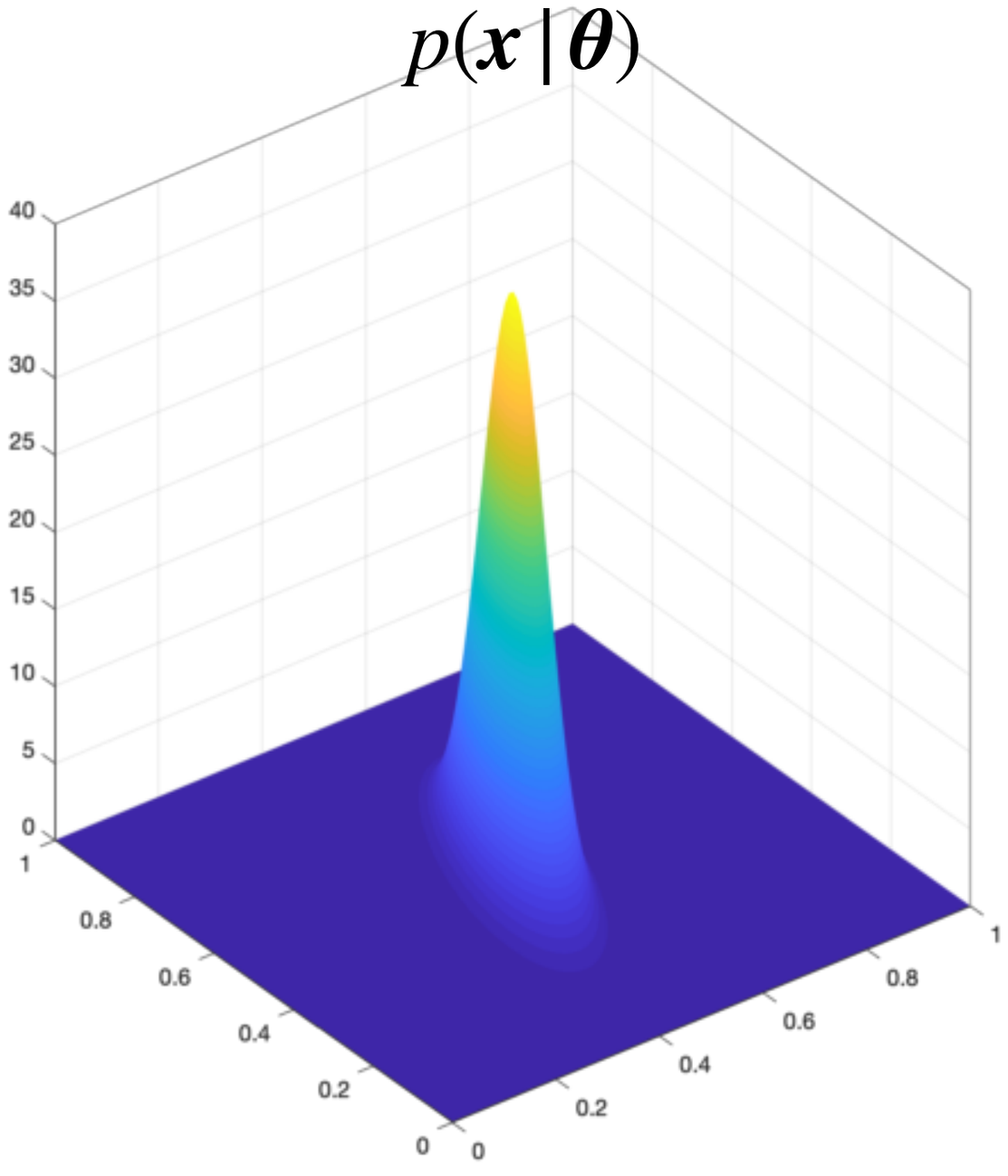
$$\boldsymbol{\Sigma} = \frac{1}{N}\sum_{n=0}^{N} (\boldsymbol{x}_n - \boldsymbol{\mu})(\boldsymbol{x}_n - \boldsymbol{\mu})'$$

Result

Top view



$p(\boldsymbol{x} \mid \boldsymbol{\theta})$



How to quantify the quality of the model?

- The goal is to have a model that captures the probability of generating or observing data x within a certain area.

- For this we can use the **Likelihood**, which characterises the probability of observing data x from a dataset $\mathcal{X}$.

$$likelihood = p(\mathcal{X}\,|\boldsymbol{\theta})$$

$\boldsymbol{\theta}$ Represents the parameters of your model (e.g., the mean and variance)

- We assume that our data is i.i.d., so :

$$p(\mathcal{X}\,|\boldsymbol{\theta}) = \prod_{n=1}^{N} p(\boldsymbol{x}_n\,|\boldsymbol{\theta})$$

- In computer science, we prefer to compute sums, and we are used to minimise quantities during optimisation processes. For this reason, we often compute the **negative-log-likelihood**:

$$\mathscr{L} = -\log p(\mathcal{X}\,|\boldsymbol{\theta}) = -\sum_{n=1}^{N} \log(p(\boldsymbol{x}_n\,|\boldsymbol{\theta}))$$

- This is analogous to the concept of loss functions.

How to quantify the quality of the model?

- This is analogous to the concept of loss functions:
  We search for the parameter values that minimise this quantity (knowing that the dataset is fixed, as the data has already been observed!).

- For a gaussian distribution: $p(\boldsymbol{x}_n | , \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}, \boldsymbol{\Sigma})$

Example with univariate gaussian:

$$\mathcal{L} = -\log p(\mathcal{X} | \boldsymbol{\theta}) = -\sum_{n=1}^{N} \log(p(\boldsymbol{x}_n | \boldsymbol{\theta})) = -\sum_{n=1}^{N} \log(\frac{1}{\sqrt{2\pi\sigma^2}} * e^{-\frac{(x-\mu)^2}{2\sigma^2}})$$

$$= \frac{N}{2}\log(2\pi) + \frac{N}{2}\log(\sigma^2) + \frac{1}{2\sigma^2}\sum_{n=1}^{N}(x_n - \mu)^2$$

- Optimum is when the derivative is equal to 0: $\frac{\partial \mathcal{L}}{\partial \mu} = 0 \quad \frac{\partial \mathcal{L}}{\partial \sigma^2} = 0$

$$\frac{\partial \mathcal{L}}{\partial \mu} = \frac{1}{\sigma^2}\sum_{n=1}^{N}(\mu - x_n) = \frac{1}{\sigma^2}(N * \mu - \sum_{n=1}^{N} x_n)$$

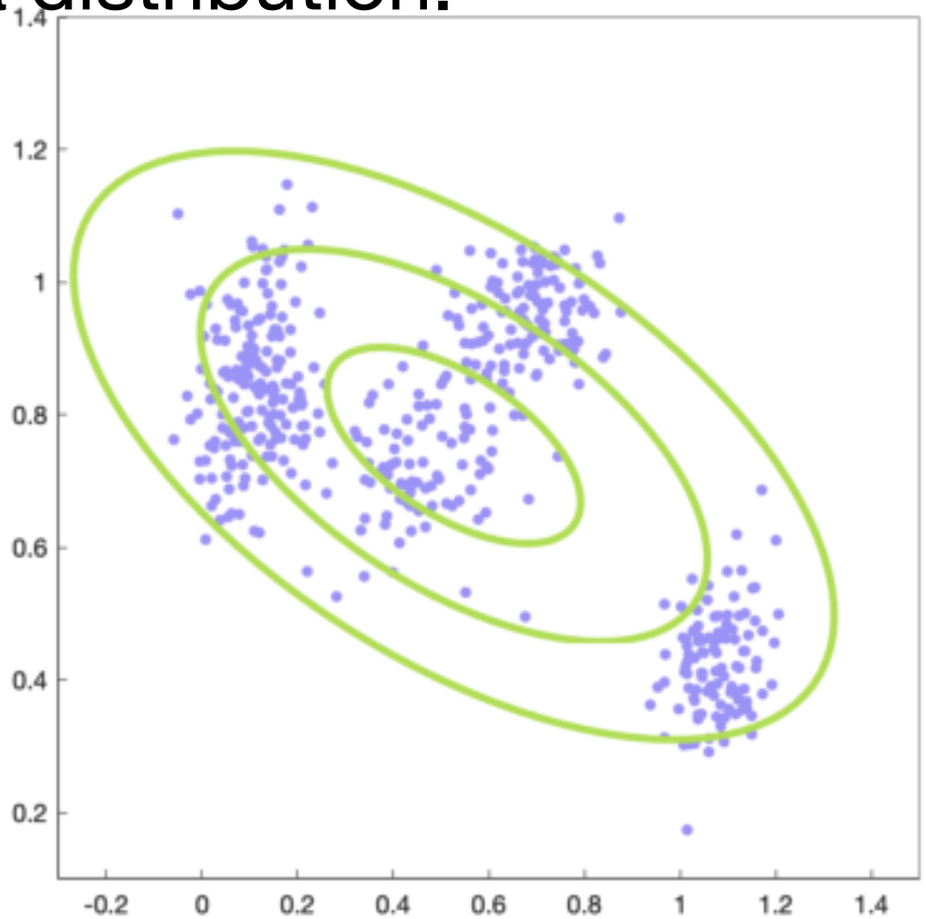
$$\frac{\partial \mathcal{L}}{\partial \mu} = 0 \rightarrow \boxed{\mu = \frac{1}{N}\sum_{n=1}^{N} x_n}$$

$$\frac{\partial \mathcal{L}}{\partial \sigma^2} = \frac{N}{2}\frac{1}{\sigma^2} - \frac{1}{2(\sigma^2)^2}\sum_{n=1}^{N}(x_n - \mu)^2$$

$$\frac{\partial \mathcal{L}}{\partial \sigma^2} = 0 \rightarrow \boxed{\sigma^2 = \frac{1}{N}\sum_{n=1}^{N}(x_n - \mu)^2}$$

Other example

- Most of the time, the gaussian distribution (or any other simple distribution) might not be sufficient to properly model the data distribution.



- **Can we use the combination (or mixture) of several distributions to construct our model?**

Mixture Models

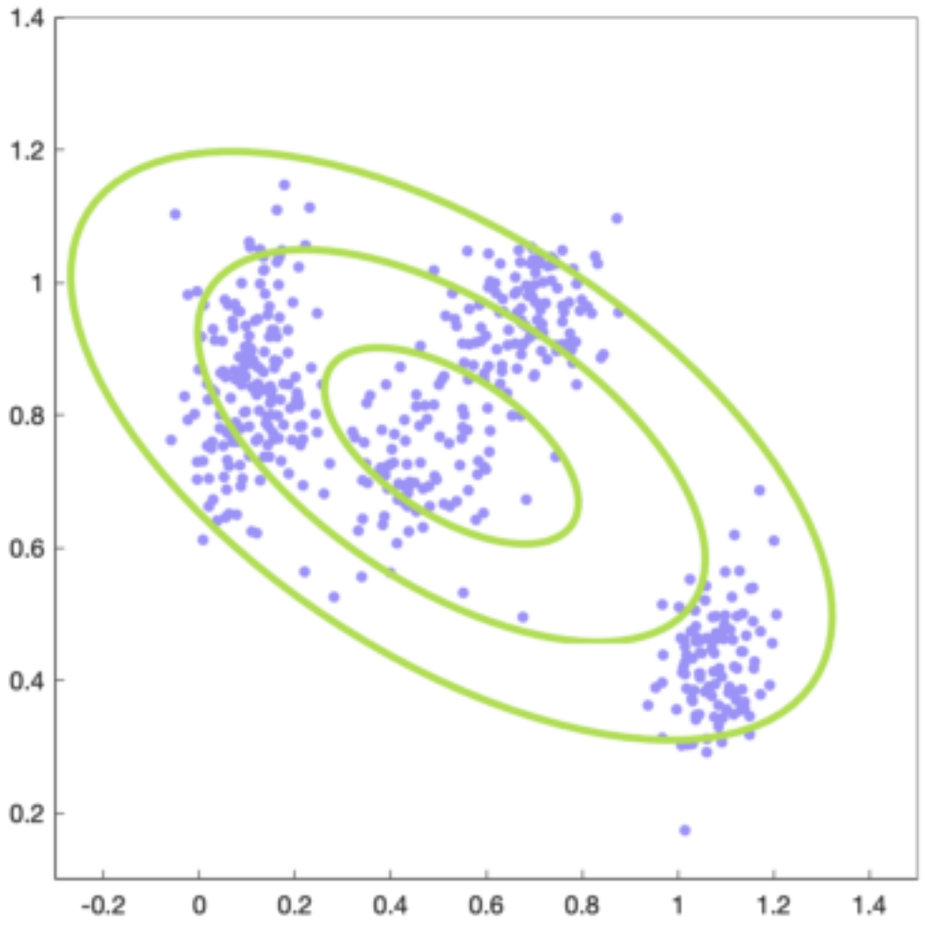
- **Yes, we can**: $p(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k p_k(\boldsymbol{x})$ $\quad 0 \leq \pi_k \leq 1 \quad \sum_{k=1}^{K} \pi_k = 1$
- We call this a mixture of models. The most famous one is **Gaussian Mixture Model** (GMM).

$$p(\boldsymbol{x} \mid \boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \qquad 0 \leq \pi_k \leq 1 \quad \sum_{k=1}^{K} \pi_k = 1$$

$$\boldsymbol{\theta} := \{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k : k = 1,...,K\}$$

Mixture Models

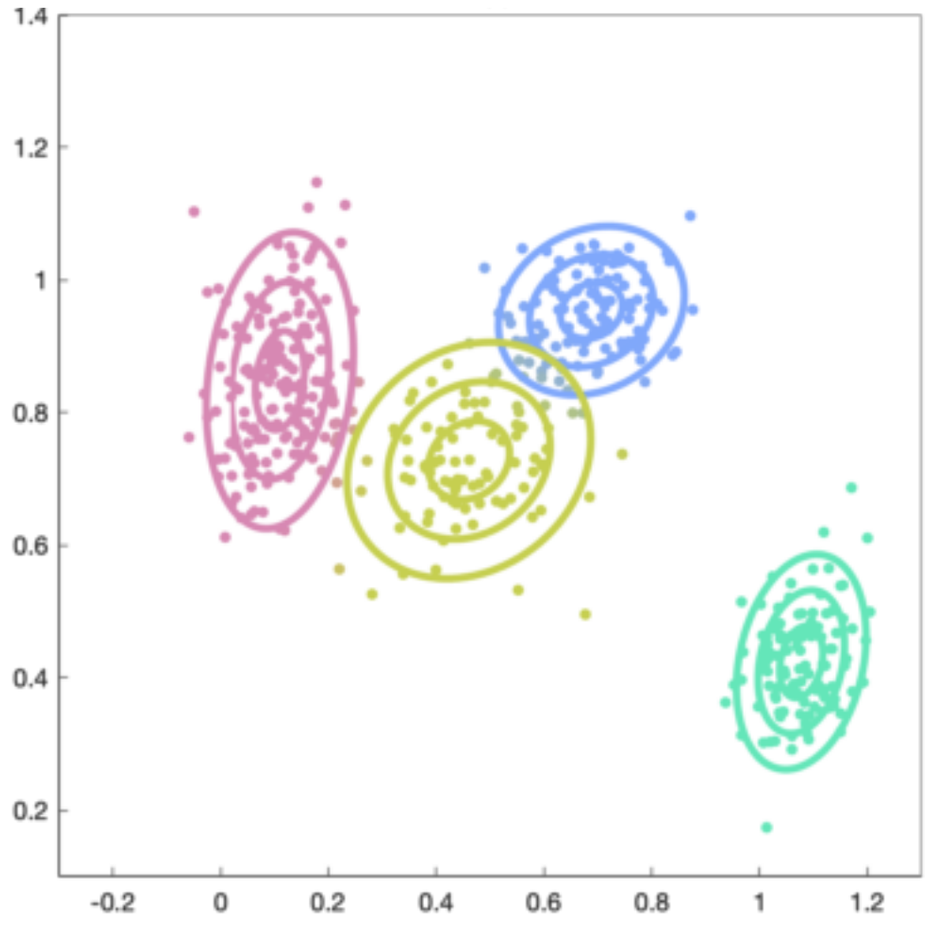
Gaussian model



$$p(\mathbf{x}\,|\,\boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}\,|\,\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

$$\boldsymbol{\theta} := \{\boldsymbol{\mu}, \boldsymbol{\Sigma}\}$$

Gaussian Mixture Model



$$p(\mathbf{x}\,|\,\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}\,|\,\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

$$0 \le \pi_k \le 1 \qquad \sum_{k=1}^{K} \pi_k = 1$$

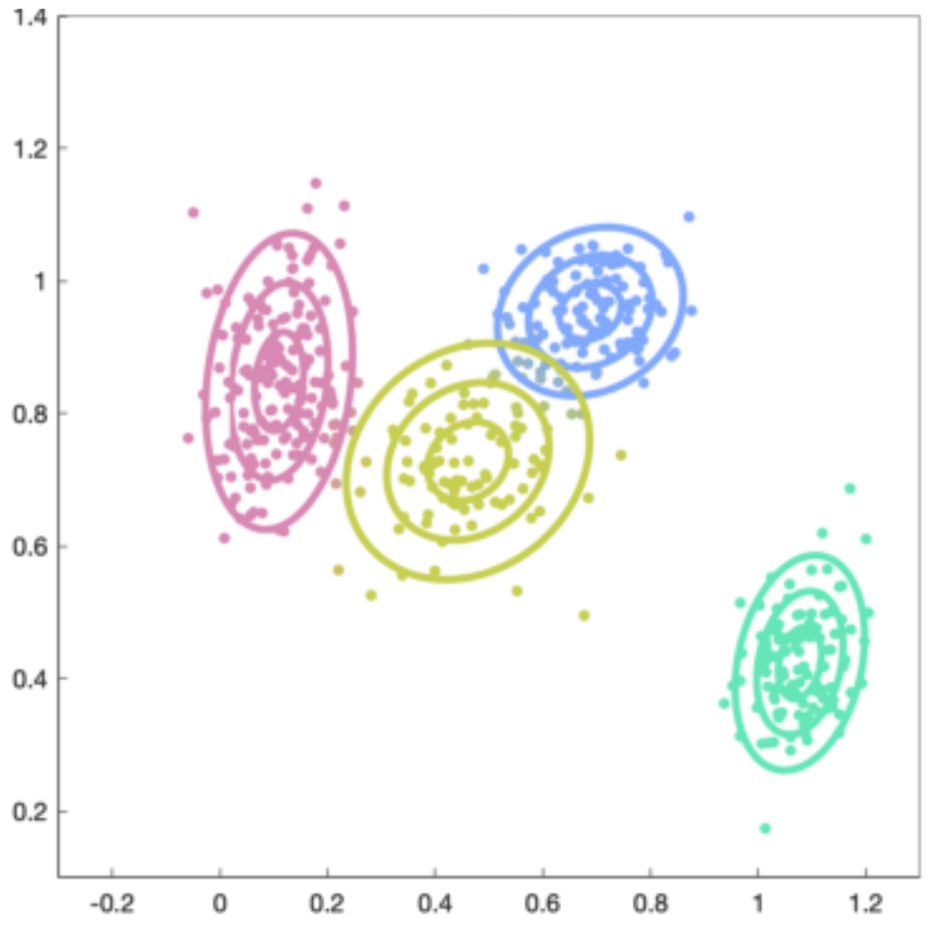
$$\boldsymbol{\theta} := \{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k : k = 1,\ldots,K\}$$

Responsibilities

- In GMM, all the k mixture components contribute to the probability distribution.

- We can compute the **responsibilities** of the *k*th mixture component for the *n*th data point:

$$r_{nk} = \frac{\pi_k \mathcal{N}(\mathbf{x}_n \,|\, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_n \,|\, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$



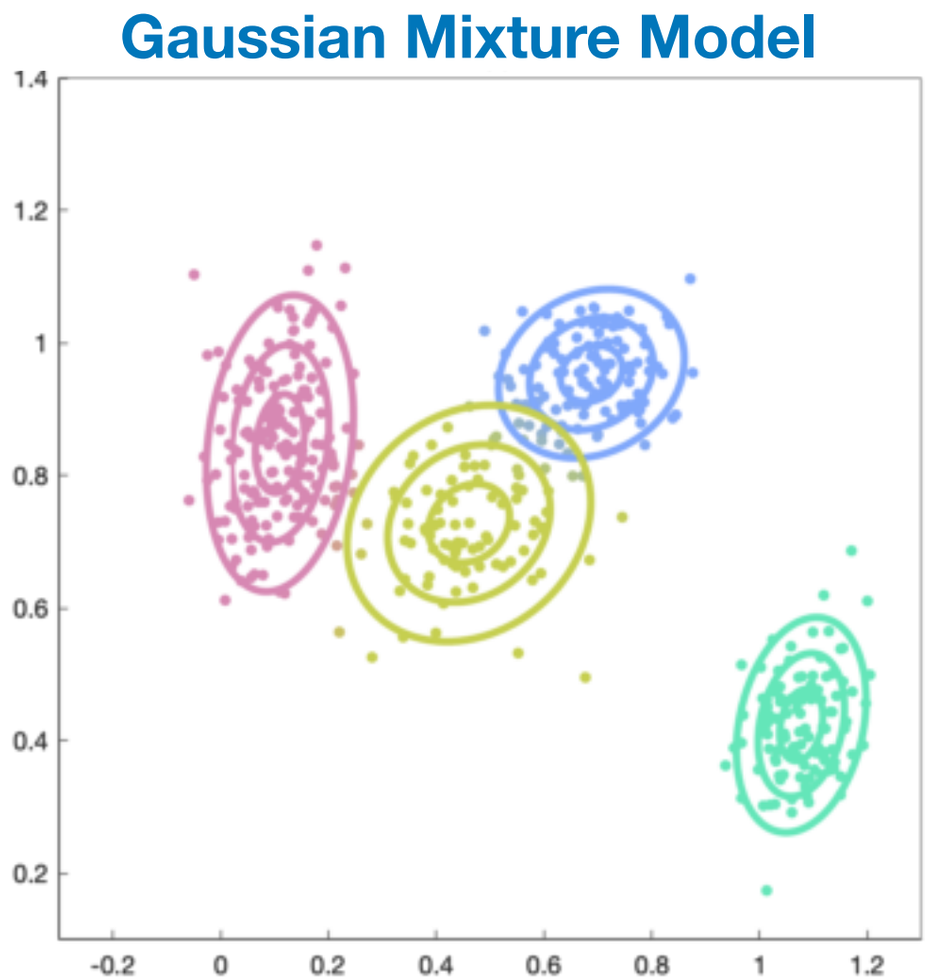
- This quantity will be very useful later in the course.

Mixture Models

- Mixture models can model more complicated data distribution, but they also have more parameters to tune!

- We have seen that the maximising the likelihood can be used to find the best parameters

- We **can't** use the same approach as before because the update of each parameter depends on the other parameters. A closed-form solution cannot be obtained.



$$p(\boldsymbol{x}\,|\,\boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}\,|\,\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

$$0 \le \pi_k \le 1 \qquad \sum_{k=1}^{K} \pi_k = 1$$

$$\boldsymbol{\theta} := \{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k : k = 1,...,K\}$$

Expectation Maximisation

- To find the parameters, we use an iterative approach called "**Expectation Maximisation**" (EM)

- EM is not restricted to GMM, and can be used with other mixture models.

- EM is composed of two steps (after the initialisation):

  - E-step: Compute the responsibilities r_{nk}
    (going futher: it corresponds to the posterior probability of data point n to belong to the mixture component k).

  - M-step: Use the updated responsibilities to re-estimate the parameters $\boldsymbol{\theta} := \{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k : k = 1,...,K\}$

- And repeat!

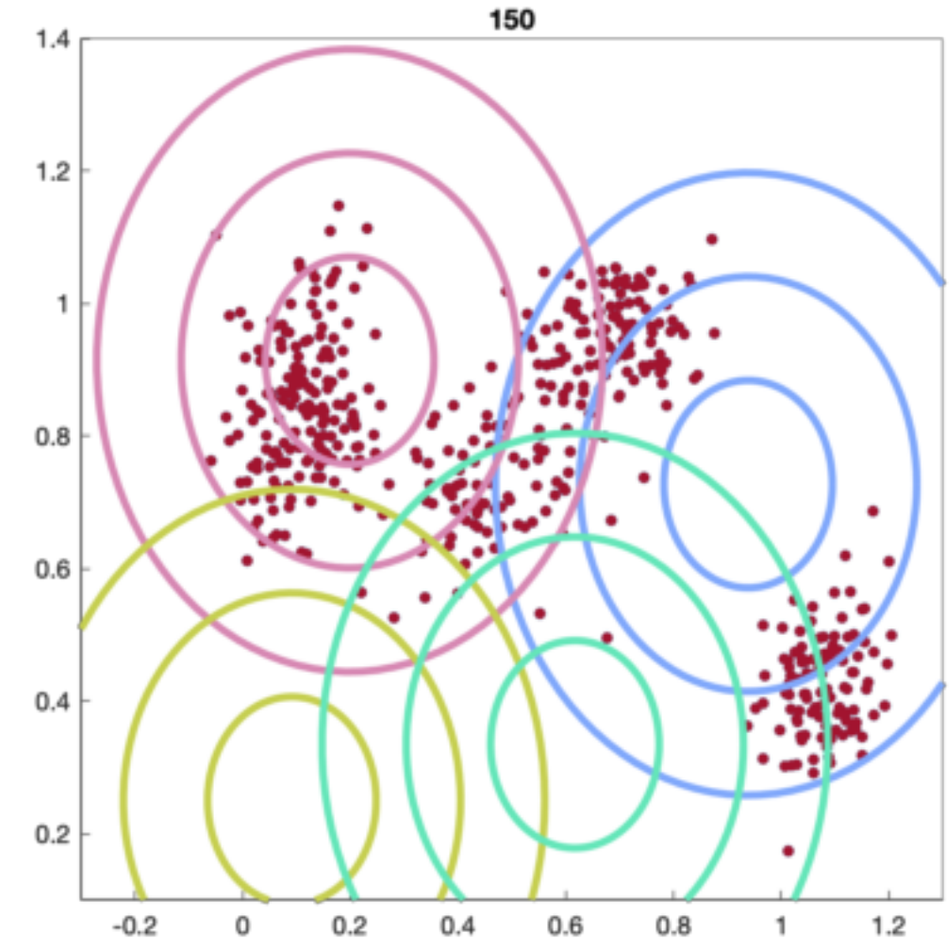
GMM-EM: Initialisation

Initialisation:

- For each mixture component, initialise

$$\boldsymbol{\theta} := \{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k : k = 1,...,K\}$$

$$\sum_{k=1}^{K} \pi_k = 1$$



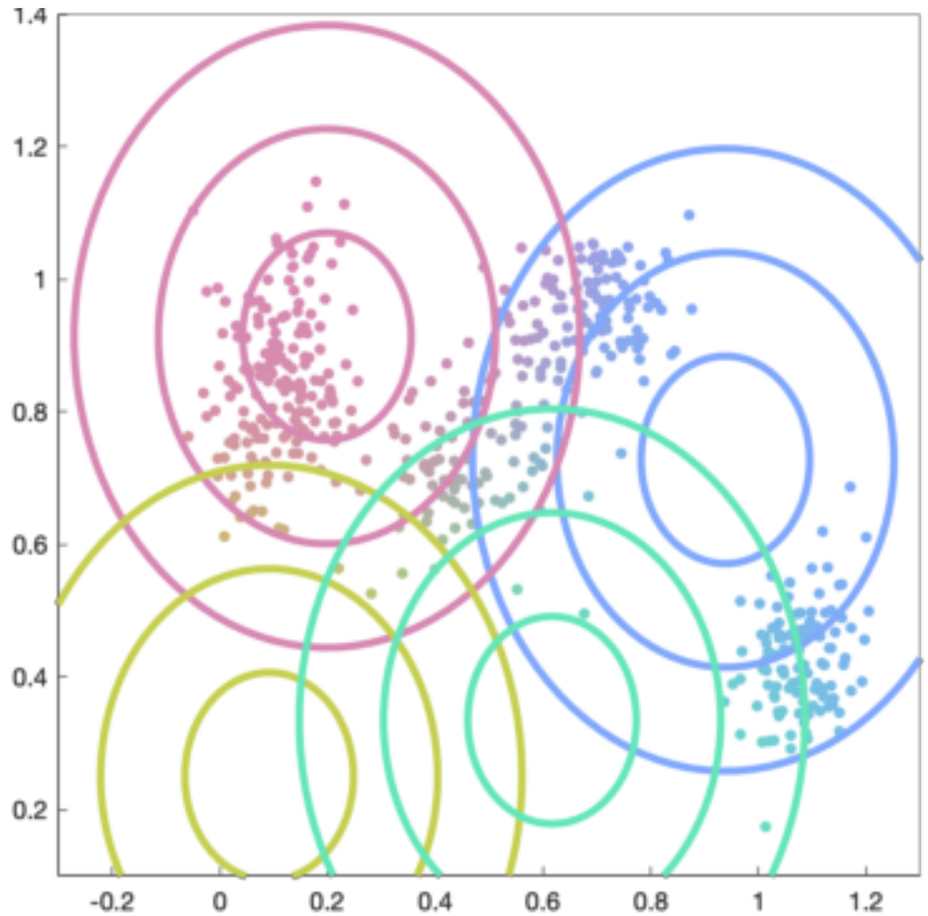
For proofs, you can refer to the Mathematics for Machine Learning book, by Deisenroth, Faisal, and Ong (chapter 11).
https://mml-book.github.io/

GMM-EM: E-Step

E-Step:

- Compute the responsibilities for each datapoint and each mixture component:

$$r_{nk} = \frac{\pi_k \mathcal{N}(\mathbf{x}_n \,|\, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_n \,|\, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$



Notice the dot colour changed

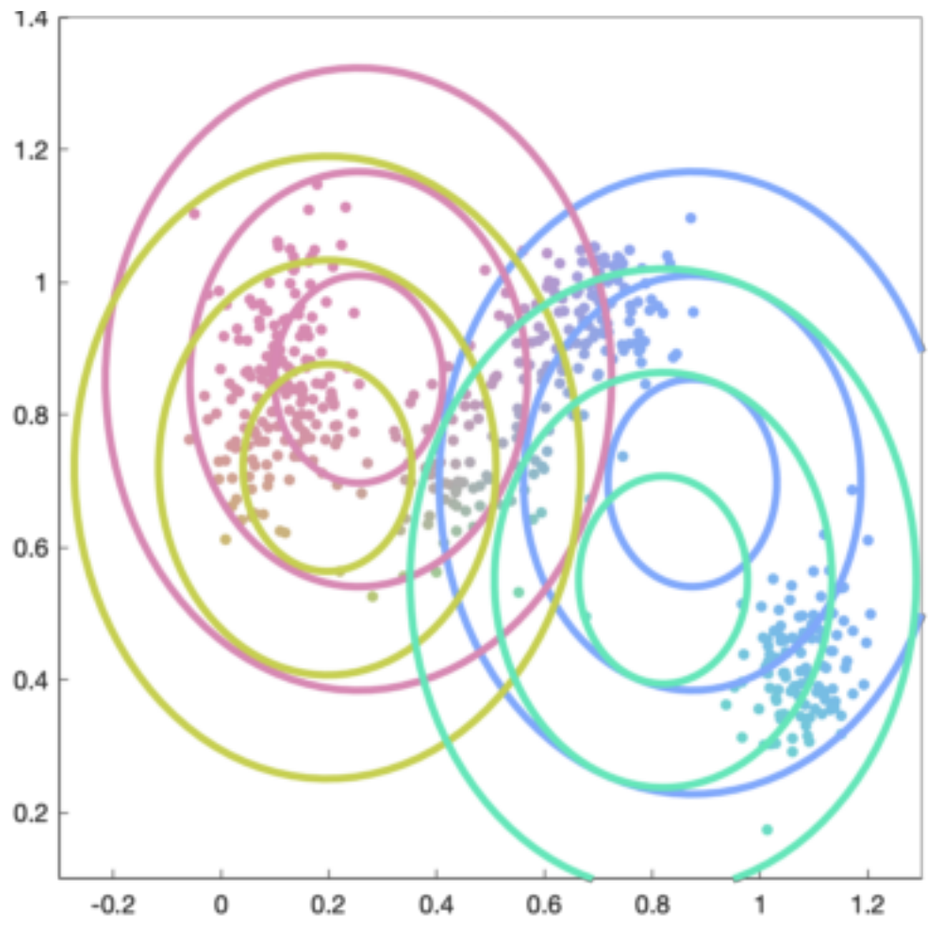
GMM-EM: M-Step (1)

M-Step:

- Update the mean of **each** mixture component:

$$\boldsymbol{\mu_k} = \frac{1}{N_k} \sum_{n=1}^{N} r_{nk} \boldsymbol{x}_n$$

with $$N_k = \sum_{n=1}^{N} r_{nk}$$



Notice the gaussians' centres moved

Going further:

- This can be interpreted as an importance-weighted Monte-Carlo estimate of the mean, where the importance weights are the responsibilities.

GMM-EM: M-Step (2)

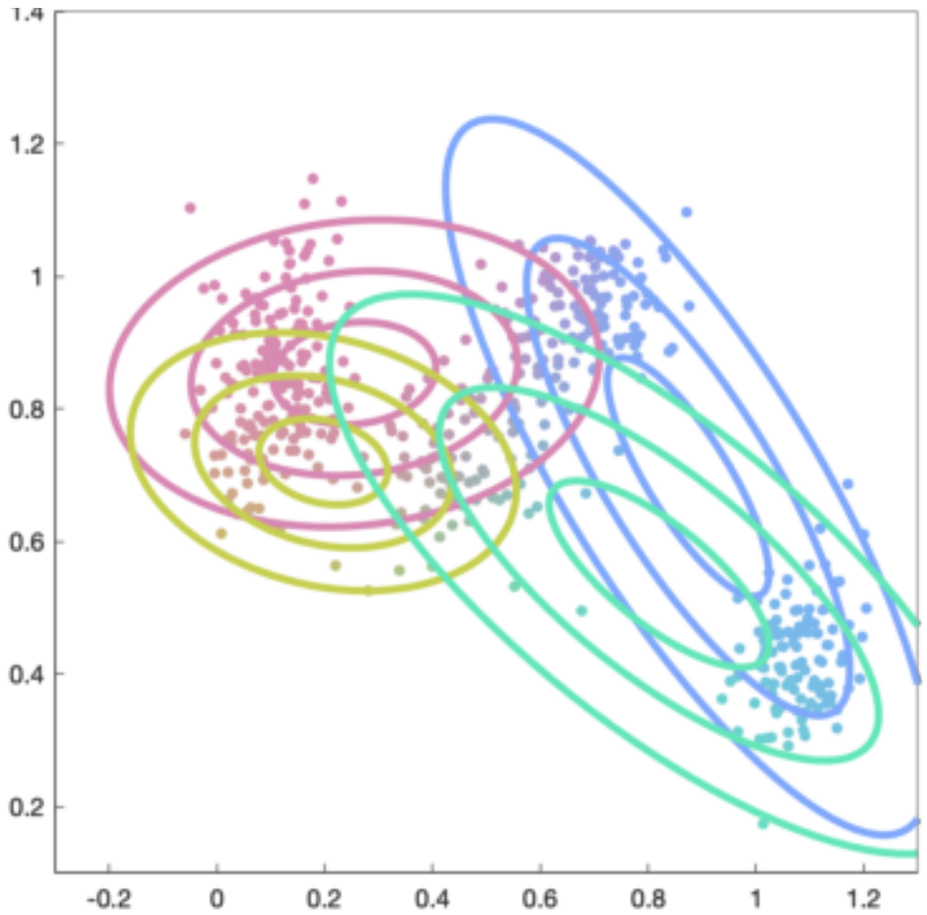
M-Step:

- Update the mean of **each** mixture component.
- Update the covariance of **each** mixture component:

$$\boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_{n=1}^{N} r_{nk}(\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)'$$

Going further:

- Similar to the mean update, this can be linked to an importance-weighted Monte-Carlo estimate.



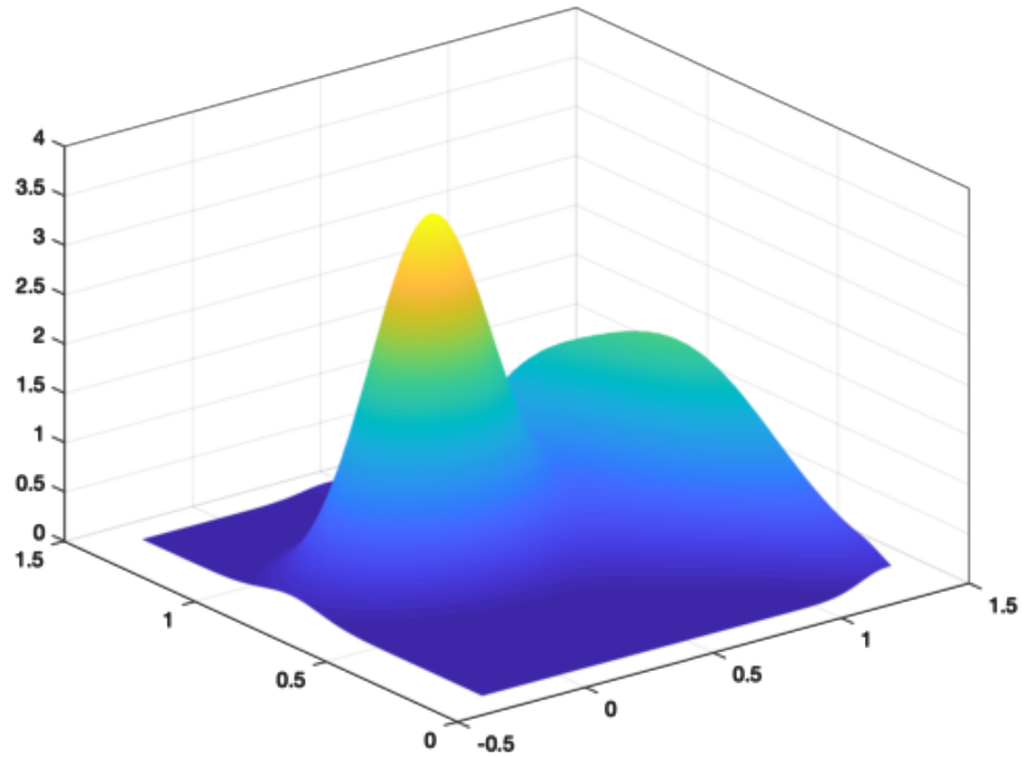
Notice shapes changed

GMM-EM: M-Step (3)

M-Step:

- Update the mean of **each** mixture component.
- Update the covariance of **each** mixture component.
- Update the weight of **each** mixture:

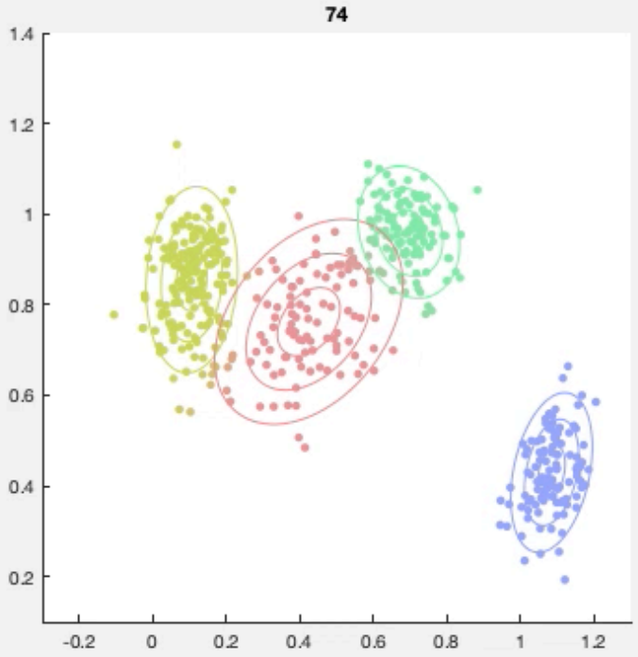
$$\pi_k = \frac{N_k}{N}$$



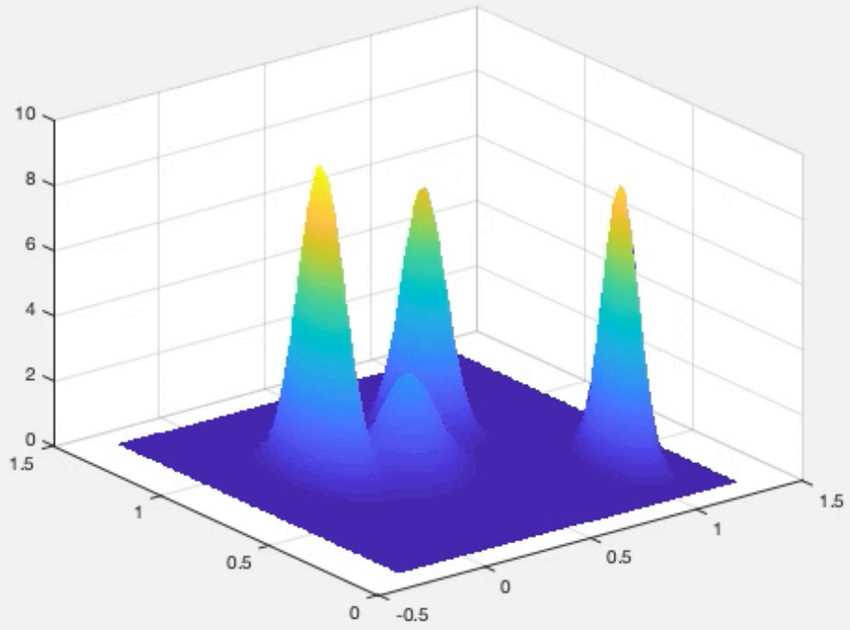
Notice: the "height" change

GMM-EM recap

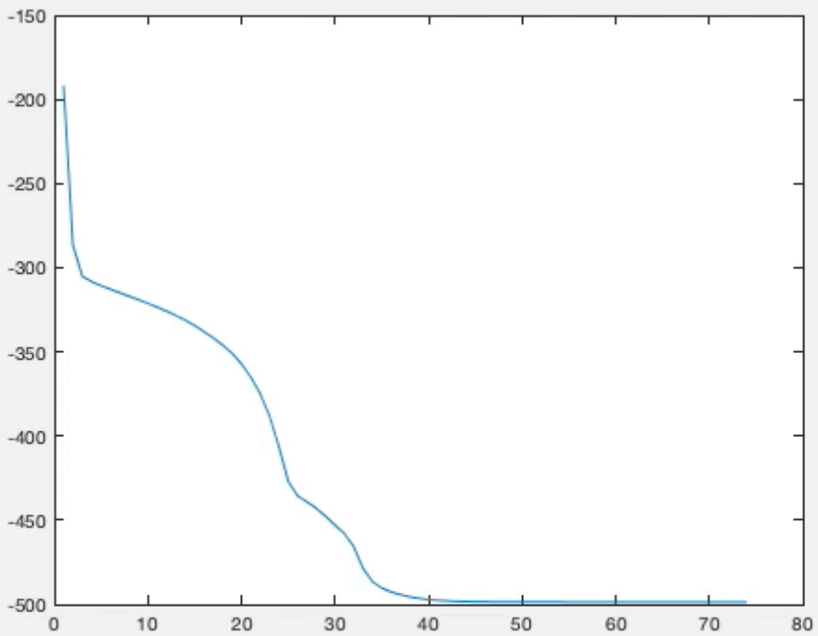
Top view



$p(\boldsymbol{x} \mid \boldsymbol{\theta})$



negative-log-likelihood



GMM-EM pseudo-code:

Initialise

E-Step:

- Compute the responsibilities

M-Step:

- Update the mean.
- Update the covariance.
- Update the weight.

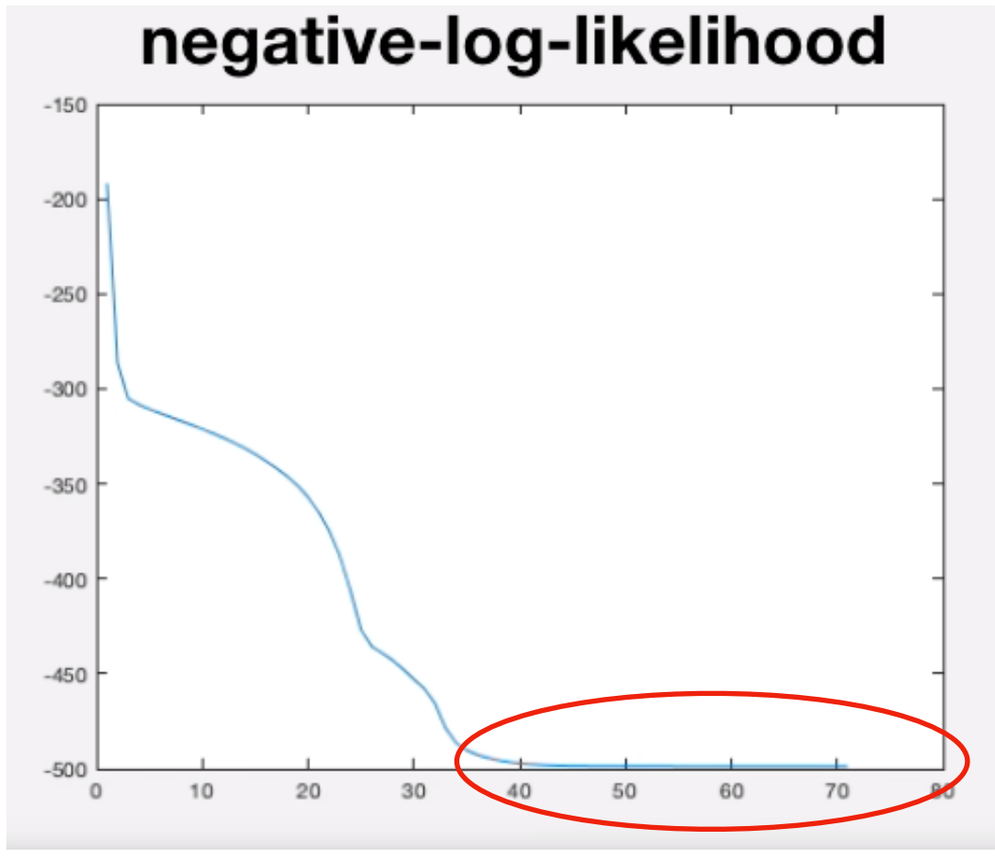
Repeat E and M steps until convergence

Convergence

- There are two ways to determine convergence:
  - No significant variation of the parameters

$$\boldsymbol{\theta} := \{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k : k = 1,...,K\}$$

- Stagnation of the likelihood:



- Converge to local optimal
  Need to restart algorithm with different initial guess of parameters (as in K-means)

Selecting the number of components

- Similarly to k-means, we have to select the right number of components, and several approaches exist for this.

- More mixture components will lead to higher likelihood (extreme case, when K = number of data points): this might not generalise well on new data.

- **Cross-validation:** Split in training and validation sets. Run GMM-EM on the training set, and evaluate the likelihood on the validation set.

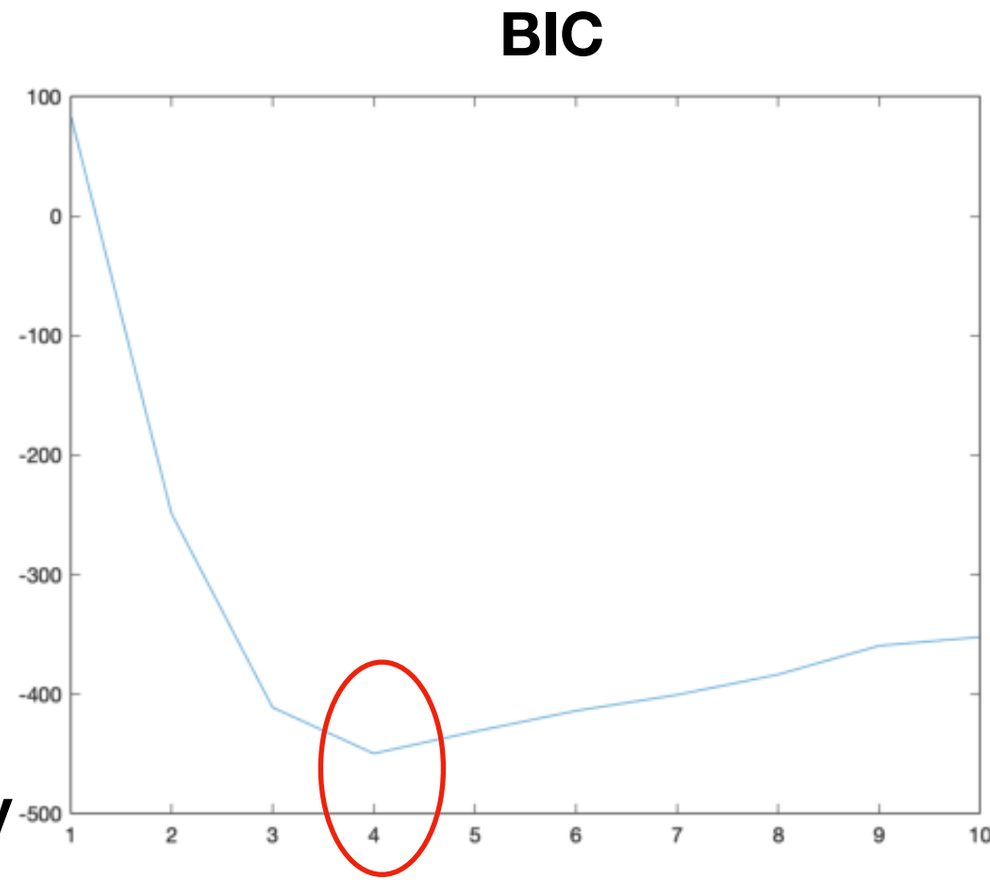
Selecting the number of components

- Occam's razor: pick the simplest of all models that fit:
  - **Bayesian Information Criterion** (BIC): (metric to be minimised)

$$BIC(K) = \mathscr{L}(K) + \frac{P(K)}{2} log(N)$$

Negative log likelihood: Encourages fitting the data

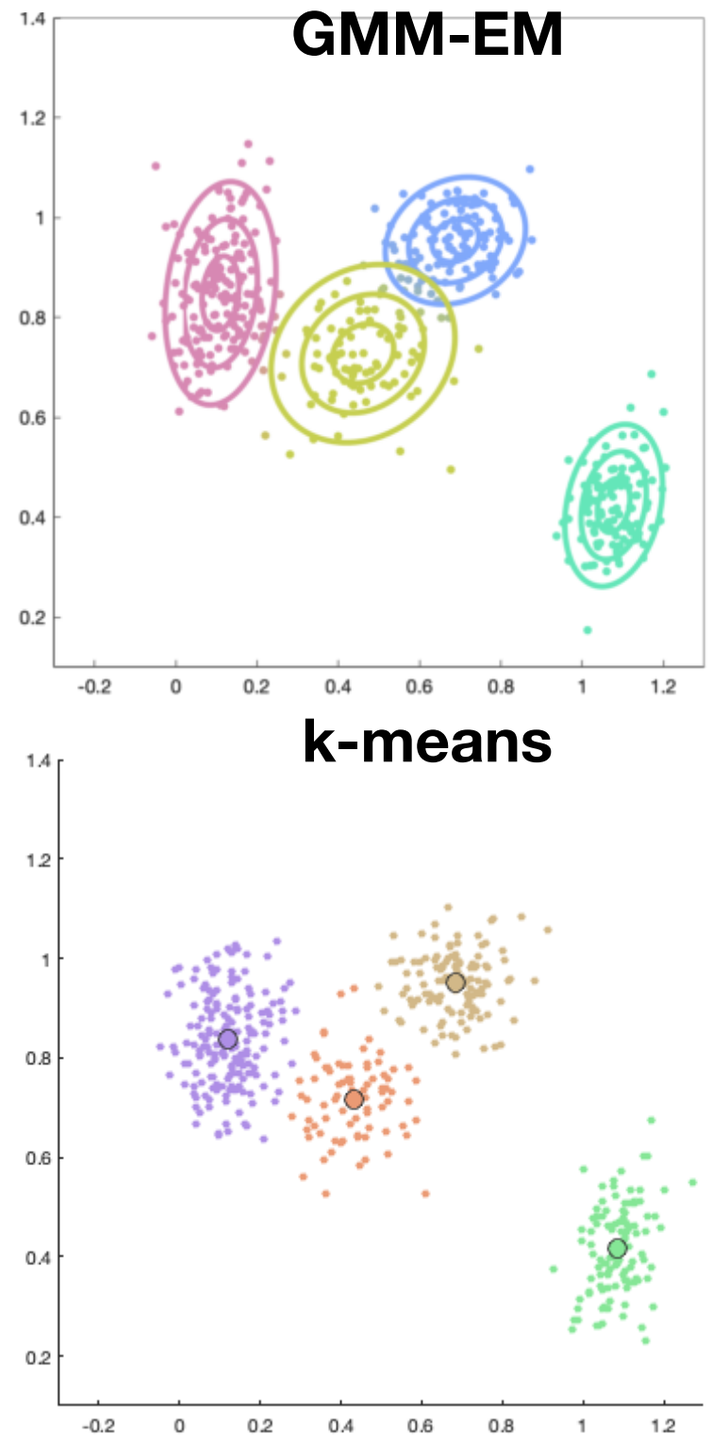
Penalises the complexity of the model



With: K = number of mixture components
N = number of data points
P(K) = number of parameters estimated by the model
For 2D gaussian: P(K)= 6*K; 2 for the mean, 3 for the covariance, 1 for the weight

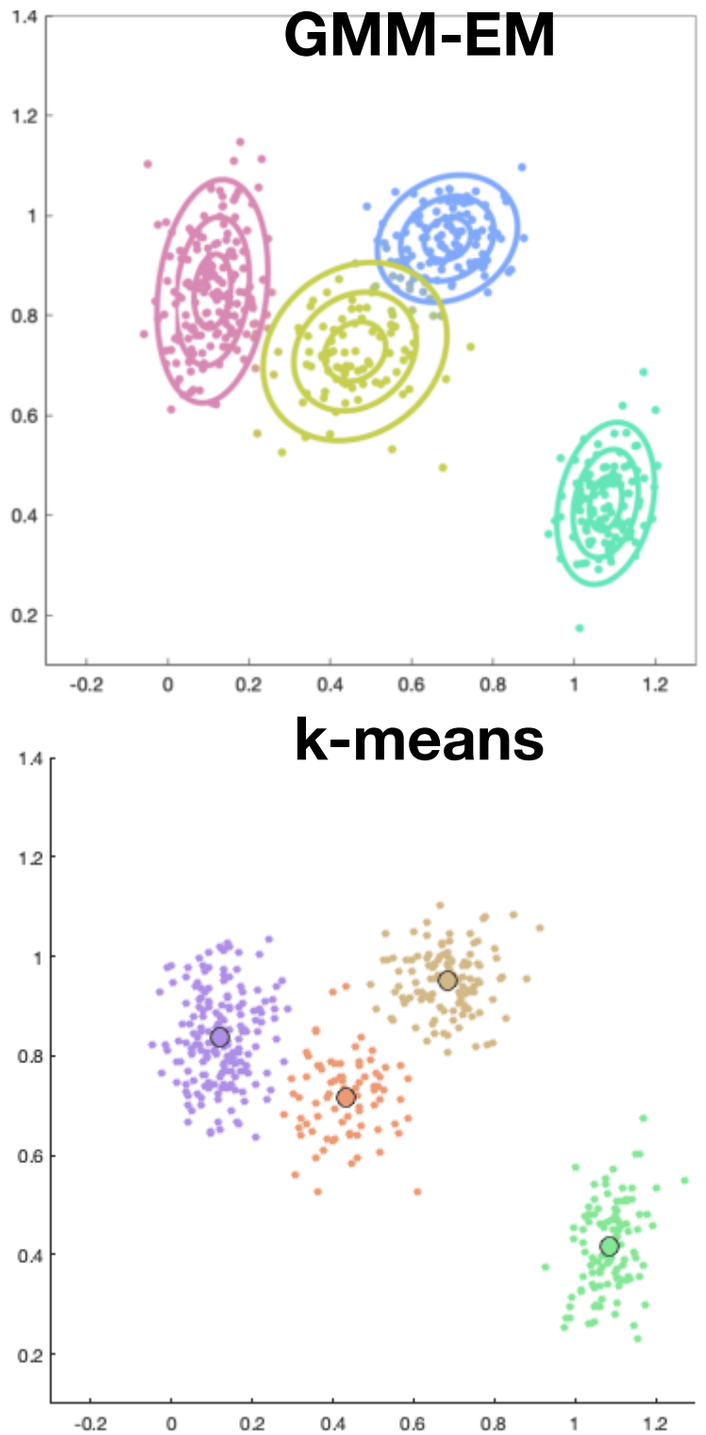
GMM-EM / k-means

- There are a lot of connections between k-means and GMM-EM.
- Similar to k-means:
  - We have to select K
- Convergence happens changes are sufficiently small
- Sensitive to initialisation.
  (We often initialise the means of GMM-EM from the result of k-means).



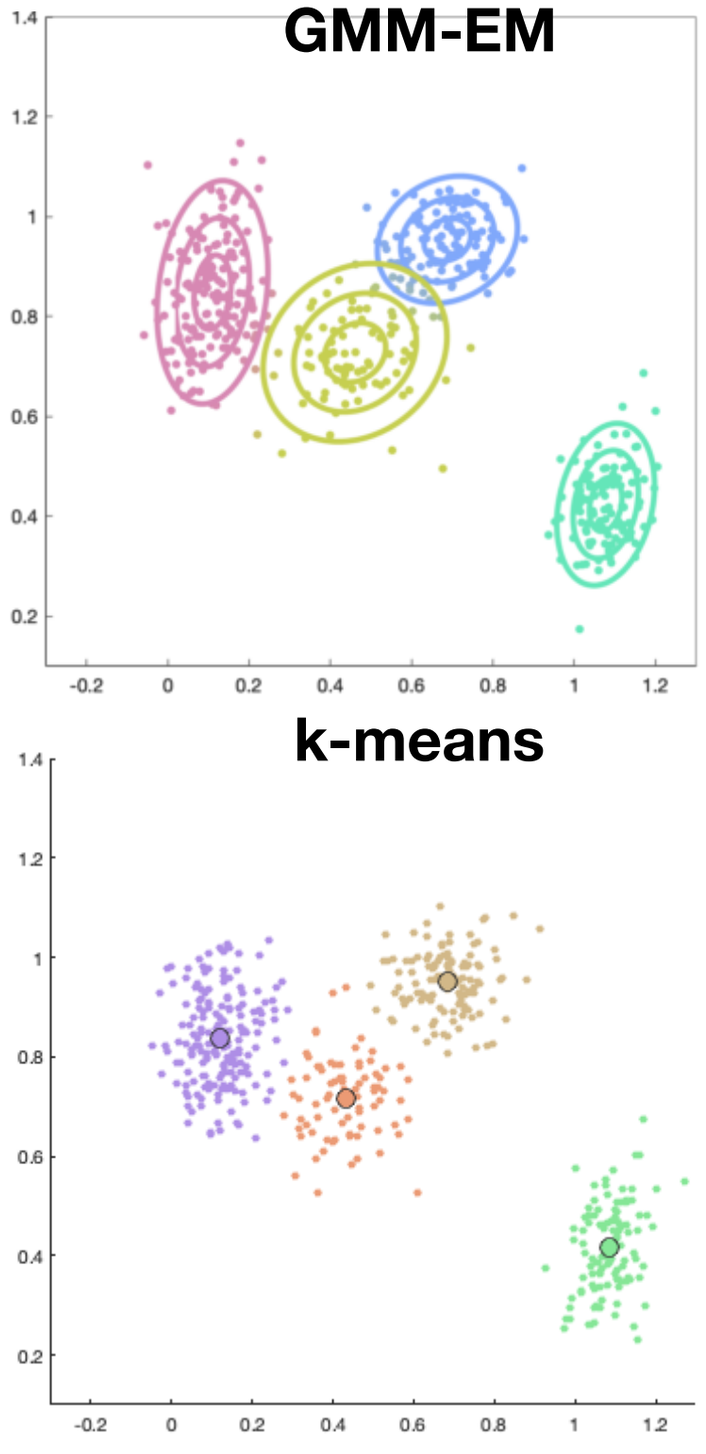
GMM-EM / k-means

- We have seen GMM-EM from the perspective of **Density estimation.**
- But like k-means, it can be used to do **unsupervised classification.**
- We can see each mixture component as a “source” that generates data points with a different probability (the weight of the GMM).



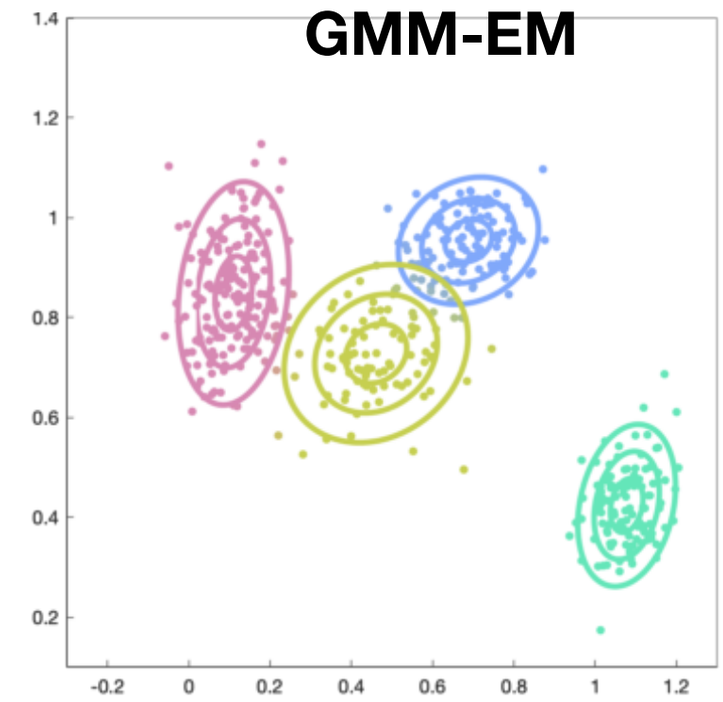
GMM-EM / k-means

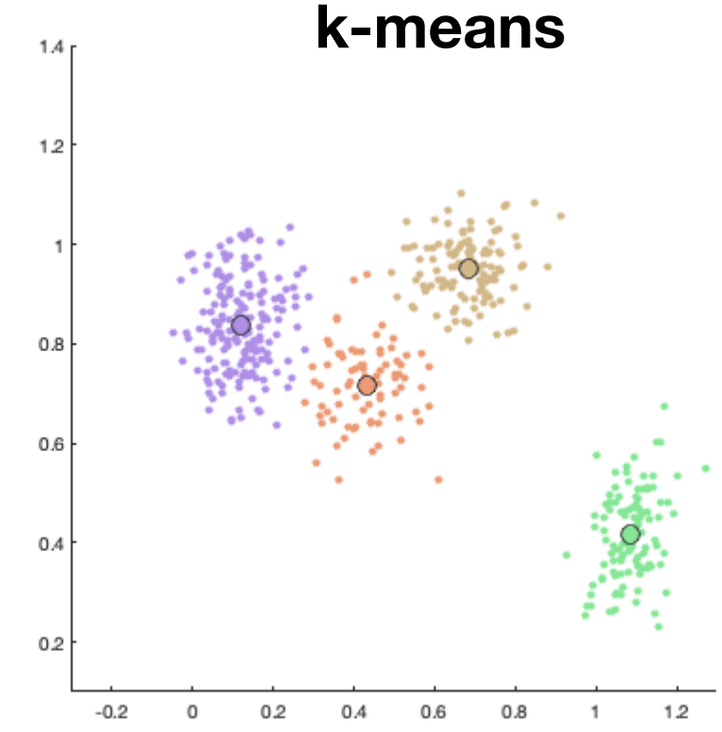
- **GMM-EM is like a “soft k-means clustering”.**
- Hard clustering (k-means): Every point belongs to exactly one cluster
- Soft clustering (GMM): Every point belongs to several clusters with certain degrees
  - Each mixture component represents a different cluster (and the responsibilities define the probability of each data point to belong to the clusters).
- **The distance to the centroids (here, the means) is not isotropic and varies during the learning process** (the distance is actually related to the Mahalanobis distance).



GMM-EM / k-means

- K-means:
  - Objective function: Minimises sum of squared Euclidean distance
  - Can be optimised by an EM algorithm
    - E-step: assign points to clusters
    - M-step: optimise clusters
  - Performs hard assignment during E-step
  - Assumes spherical clusters with equal probability of a cluster
- GMM-EM:
  - Objective function: Maximise log-likelihood
  - EM algorithm
    - E-step: Compute posterior probability of membership
    - M-step: Optimise parameters
  - Perform soft assignment during E-step
  - Can be used for non-spherical clusters
  - Can generate clusters with different probabilities





Demo

- Please fill this 2-question survey:

https://bit.ly/2BIqJeg

Exercise:

- Implement k-means/GMM-EM in matlab/python and try to reproduce the example of this couse.