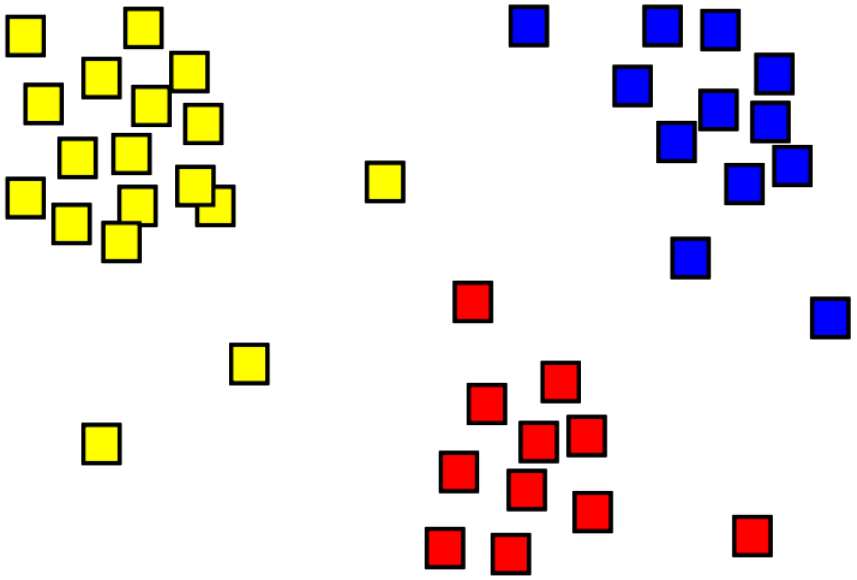
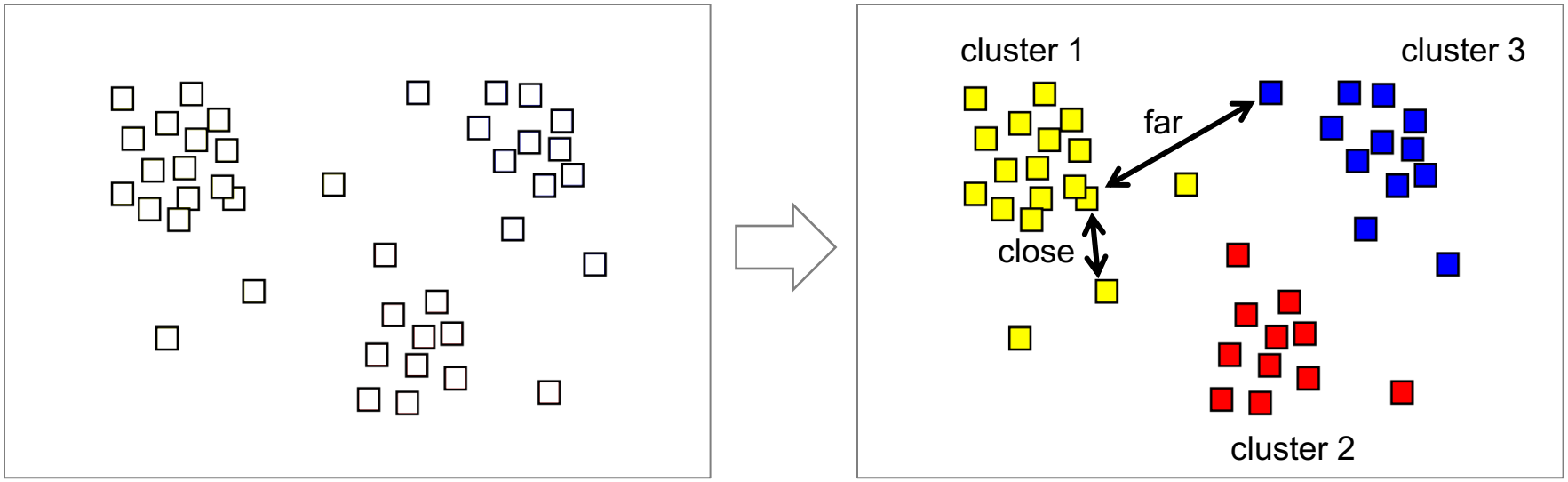
Clustering



Clustering

- Input: an "unlabeled" data set $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ where $\mathbf{x}_i \in \mathbb{R}^D$ exist in some *D*-dimensional space
- Goal: *group* the data into "clusters" based on some measure of *distance* such as $\|\mathbf{x} - \mathbf{y}\|^2$



Why cluster?

- To *identify* and *characterize* group structure in the data, for sake of *summarization* or *association*
  - Summarize: “there are *K* groups in the data and group *k* tends to have *these* characteristics”
  - Associate: “this new data point $\mathbf{x}$ belongs to group *k*”
- Examples:
  - Optimal placement of facilities / services
  - Quantization and compression
  - Grouping documents / images
  - Identifying types of customers
  - Identifying types of cells
  - Novelty detection
  - …

Quantization and compression

2 clusters

1 bit per pixel

3 clusters

~1.6 bits per pixel

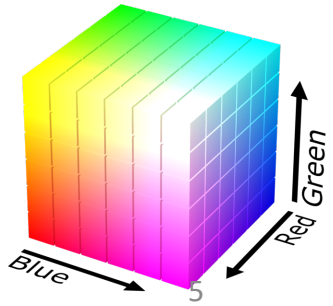
10 clusters

~3.3 bits per pixel

Original image

24 bits per pixel

$$x_{ij} = \text{intensity of colour channel } j \text{ for pixel } i$$
$$j \in \{\text{red}, \text{green}, \text{blue}\}$$



Simple kind of “lossy compression”

Image credit: Christopher M. Bishop

Grouping documents or images

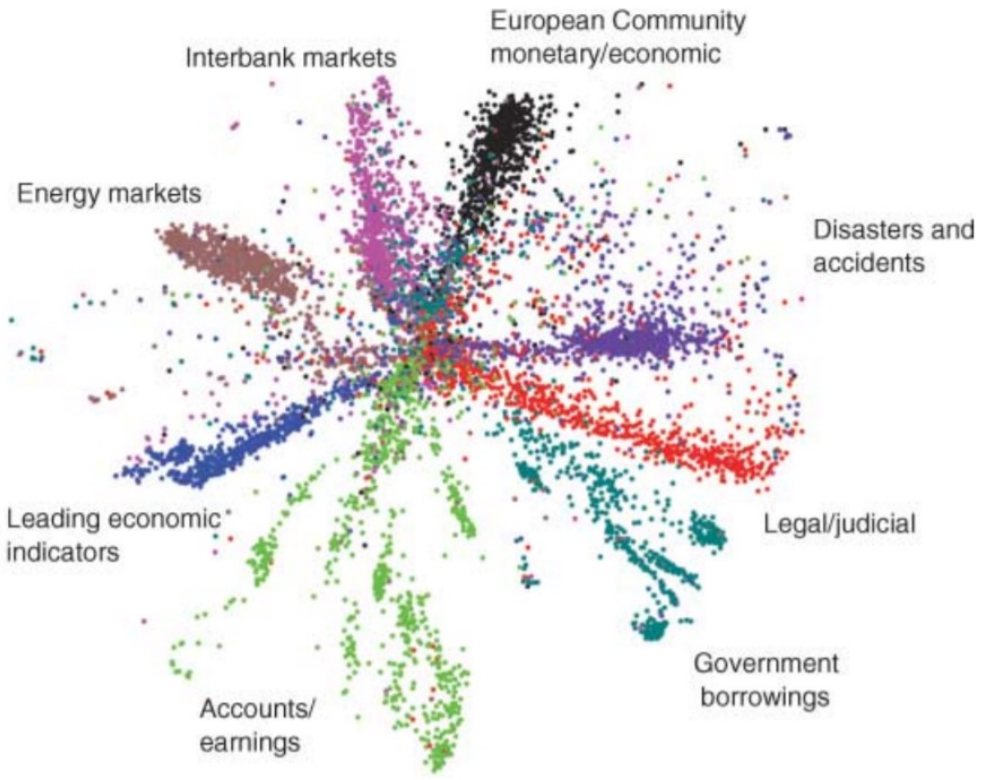
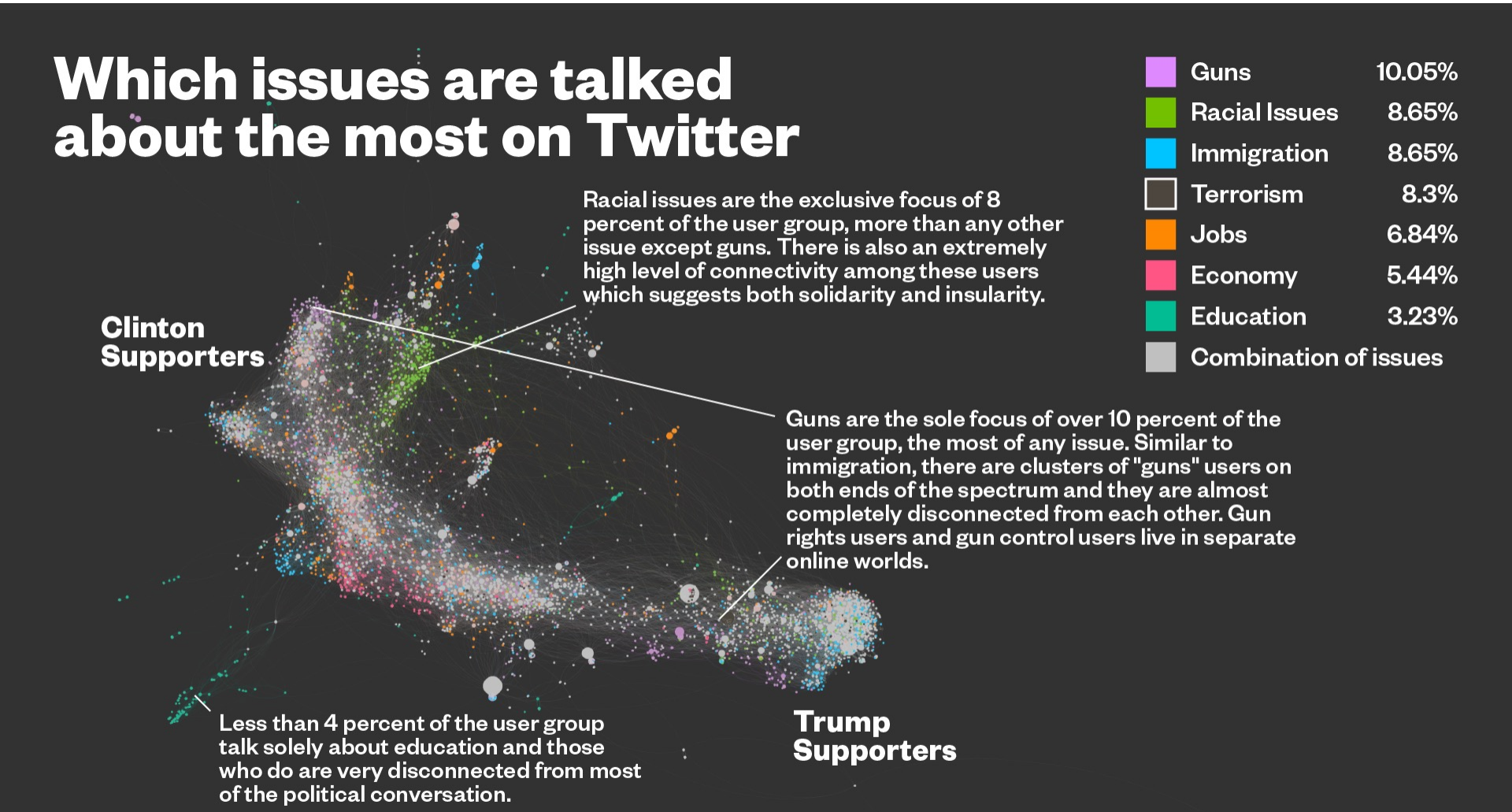


Image credit: Hinton and Salakhutdinov

Grouping text by topic / sentiment



Many cells, but how many cell *types*?

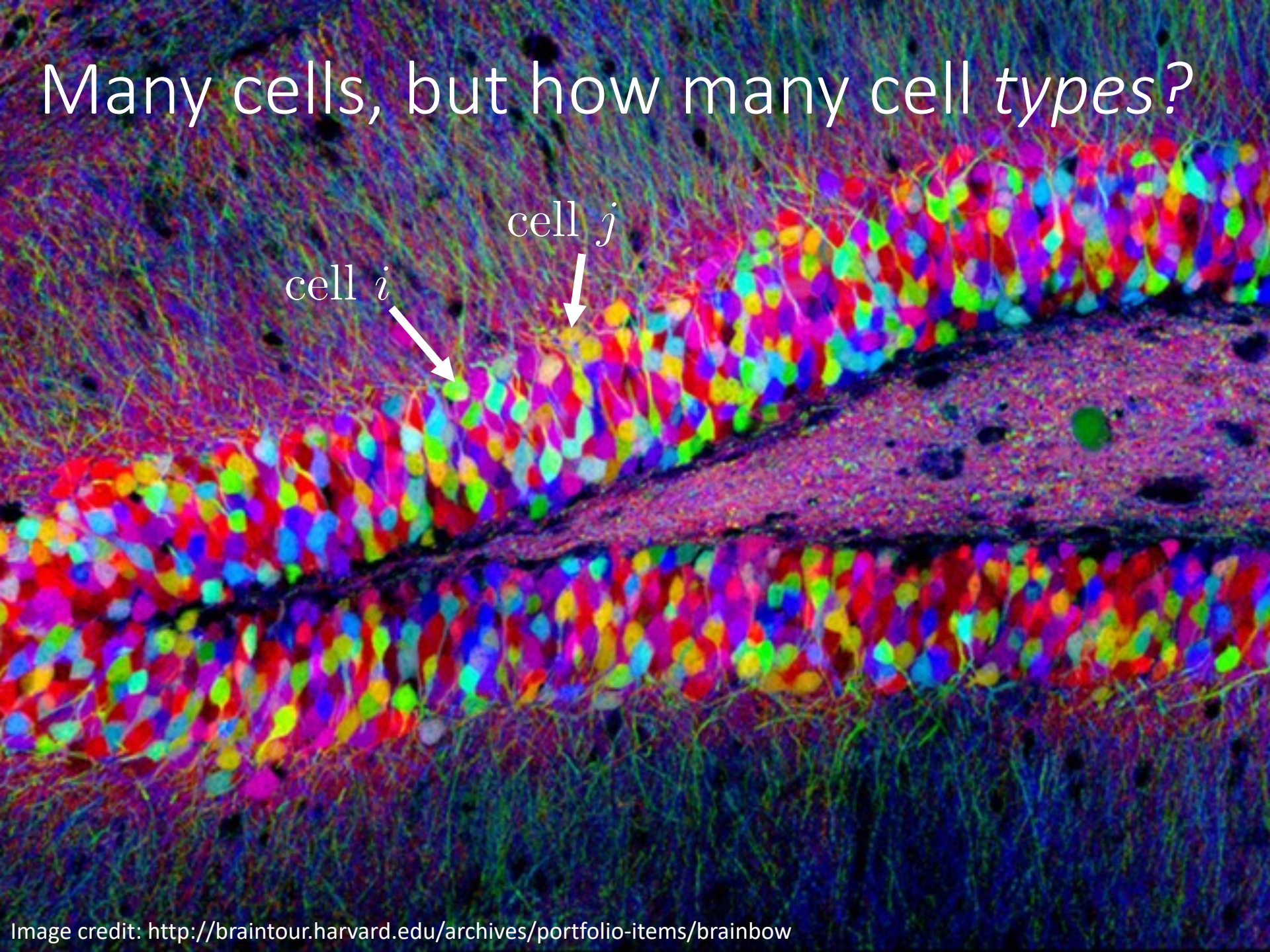
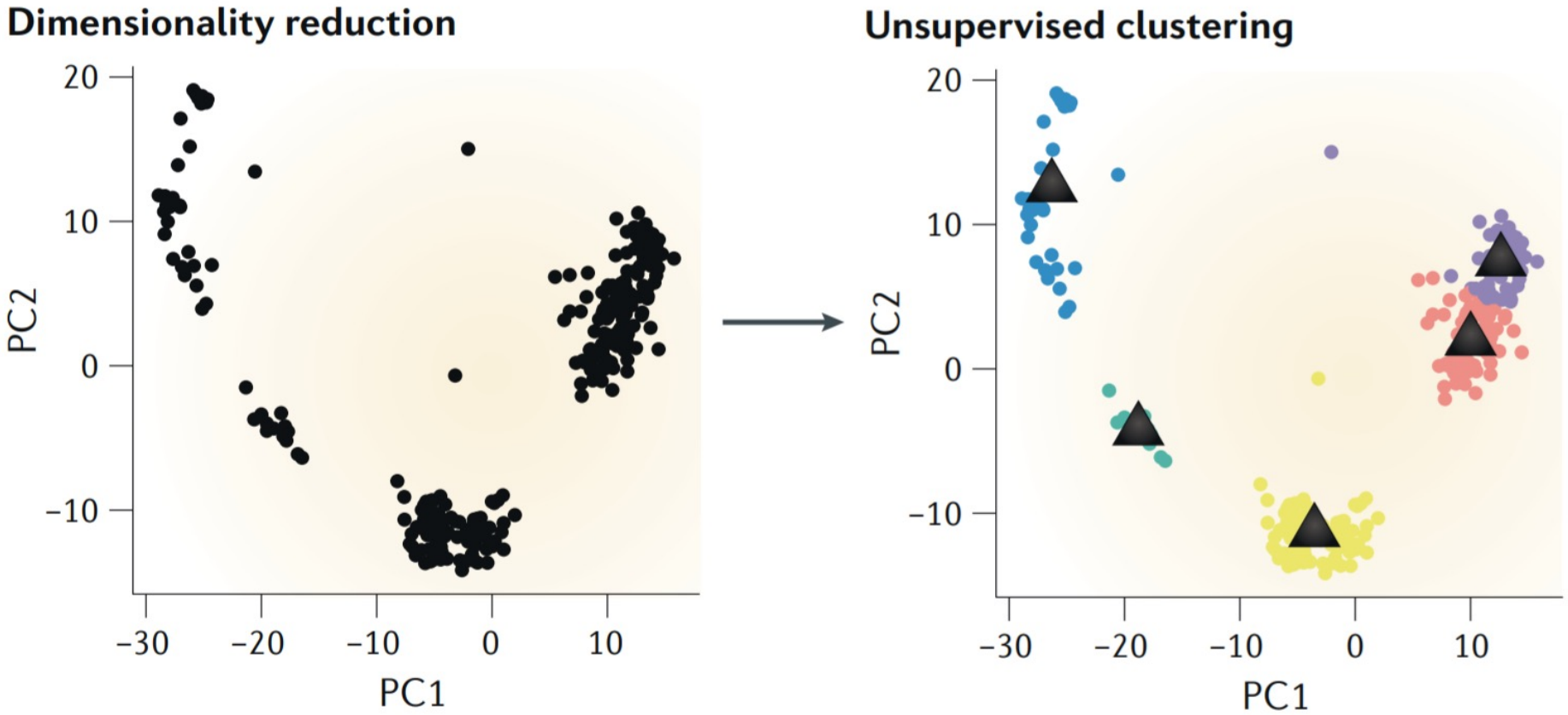


Image credit: http://braintour.harvard.edu/archives/portfolio-items/brainbow

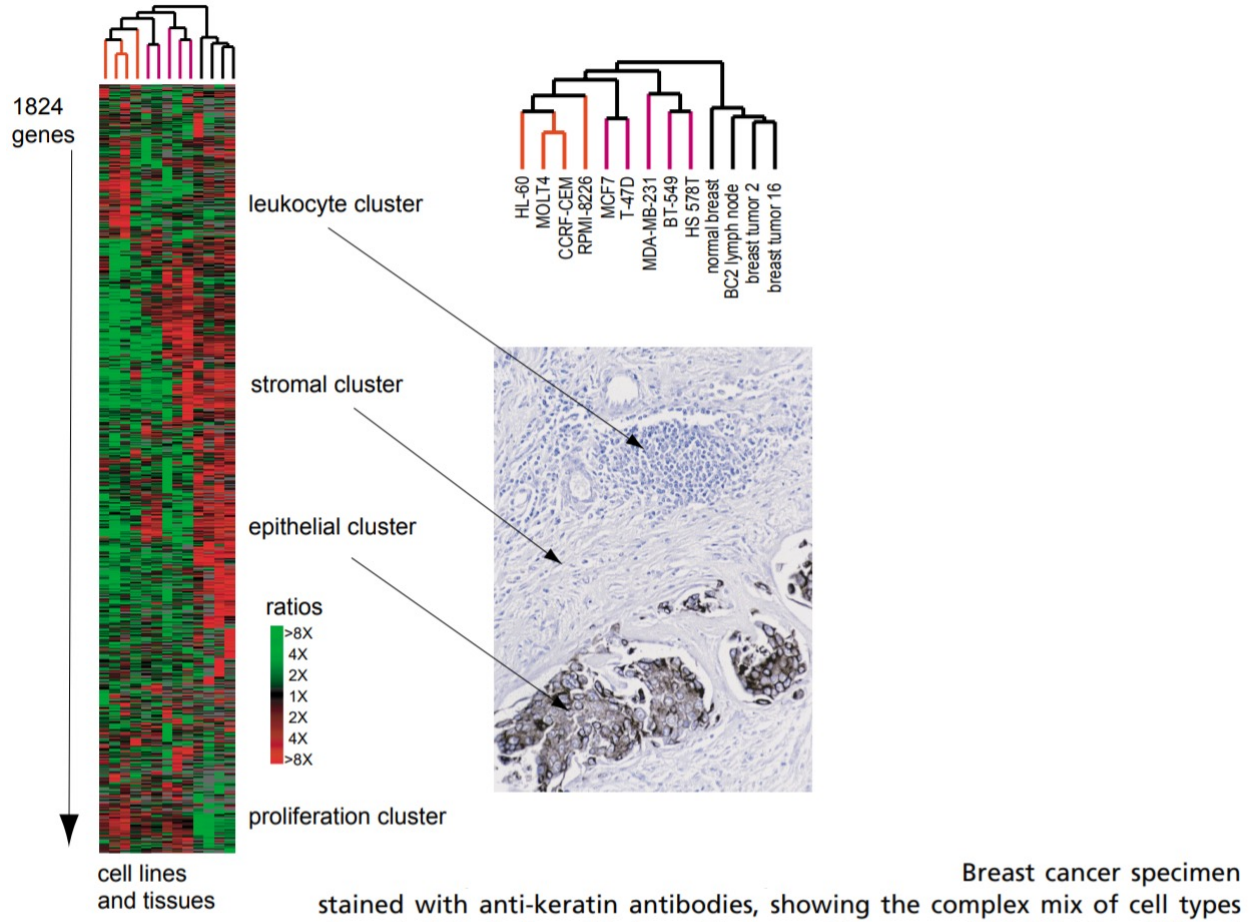
Clustering cells by gene activity



$$x_{ij} = \text{activity of gene } j \text{ in cell } i$$

Kislev et al 2019 https://www.nature.com/articles/s41576-018-0088-9

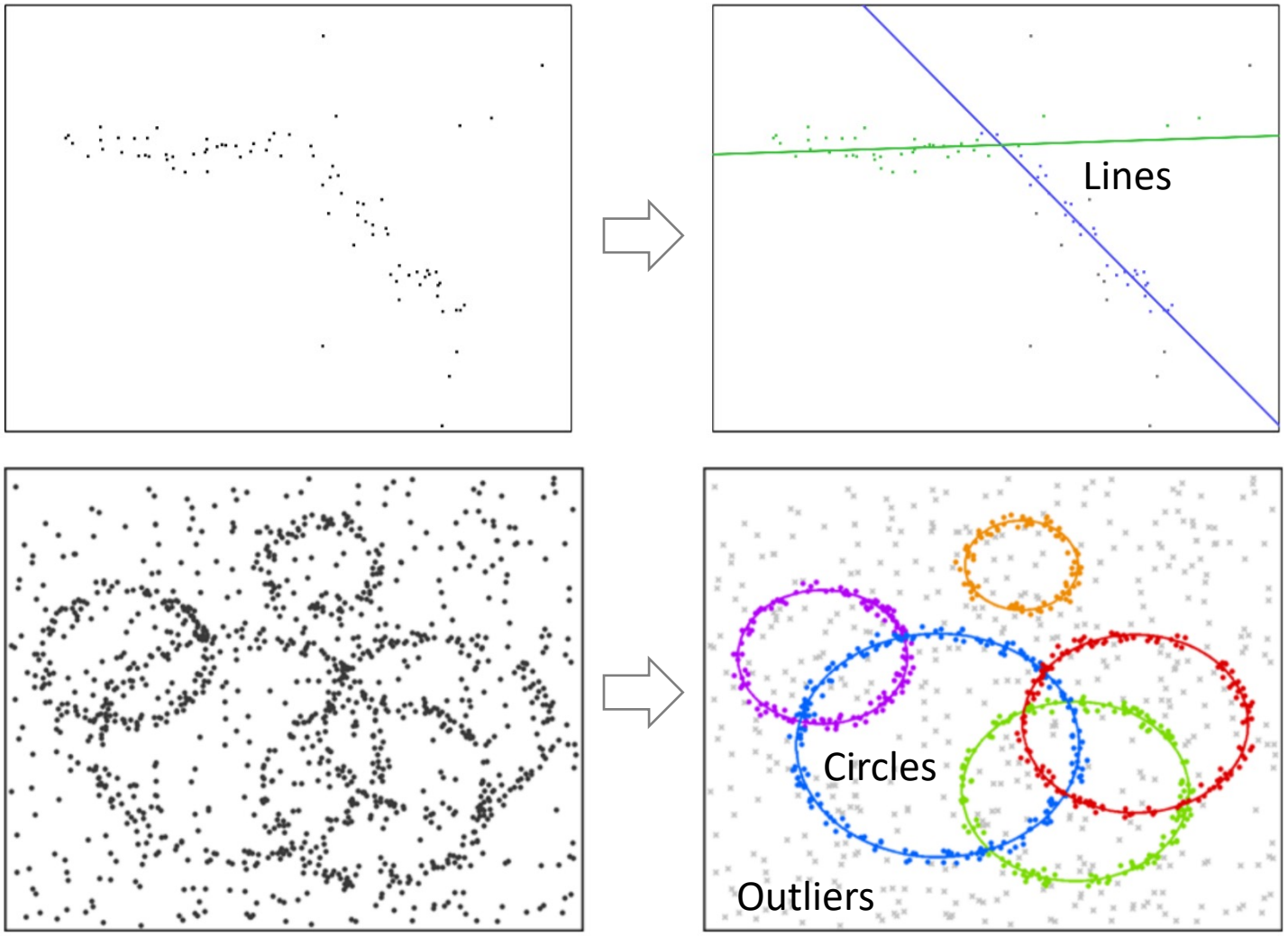
Analysis of cancer subtypes



Breast cancer specimen stained with anti-keratin antibodies, showing the complex mix of cell types characteristically found in breast tumours. The arrows highlight the different cellular components of this tissue specimen that were distinguished by the gene expression cluster analysis

Ross et al 2000 https://www.nature.com/articles/ng0300_227

Fitting geometric models to data

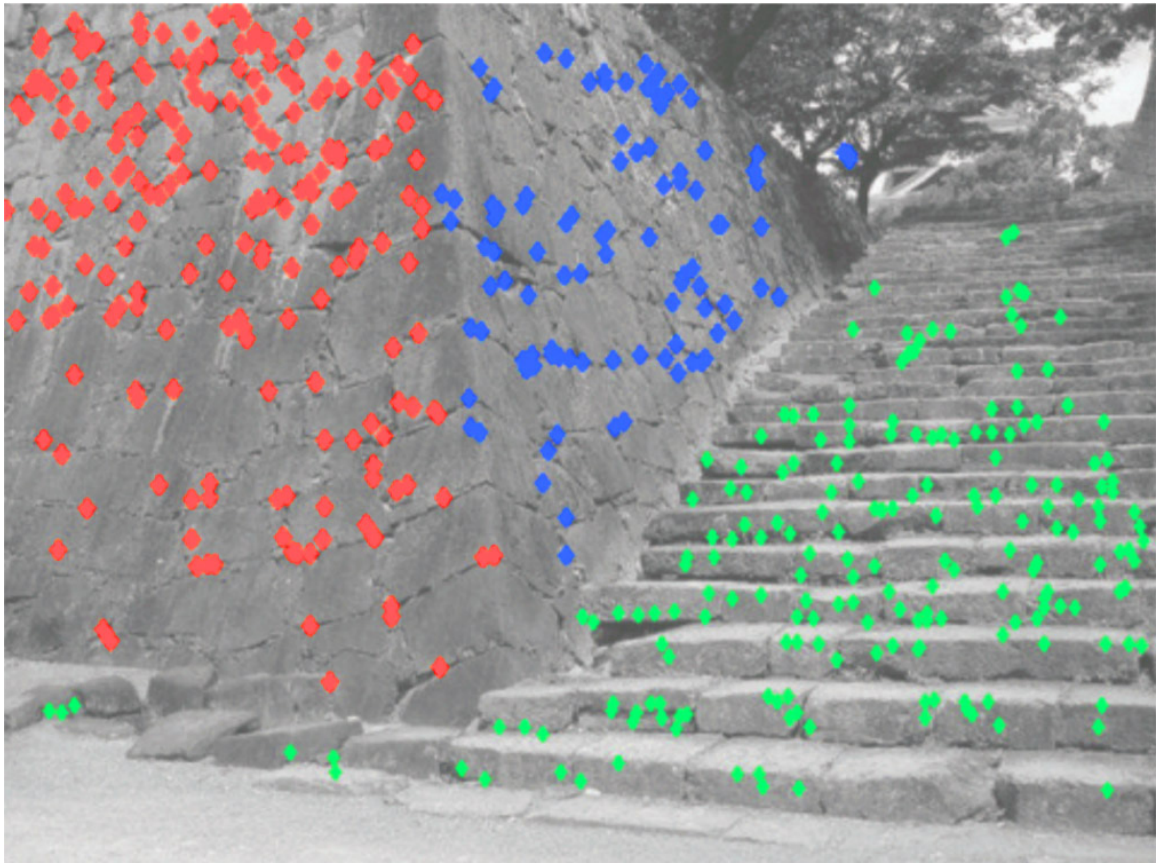


Delong et al 2012 😬

Fitting geometric models to data

Find surfaces in a 3D point cloud that was generated by a stereo camera setup.

If points can be grouped such that a 3D surface passes through them, then that's evidence of a real surface in the environment.

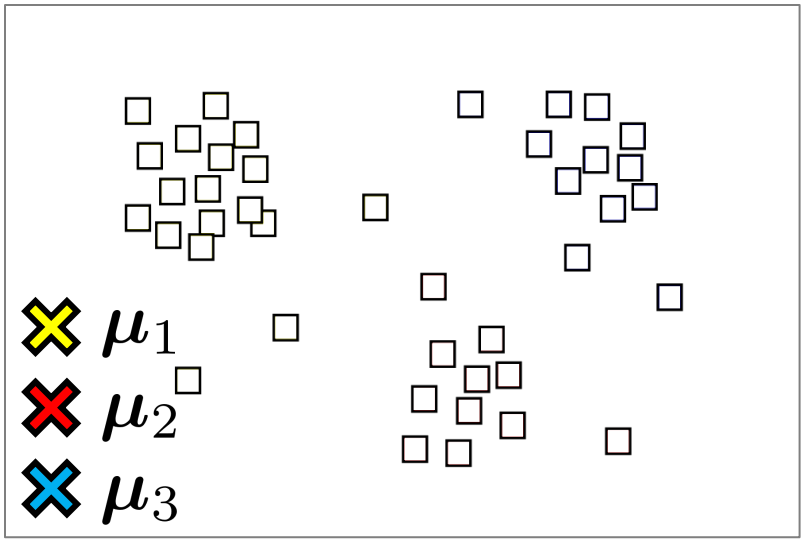


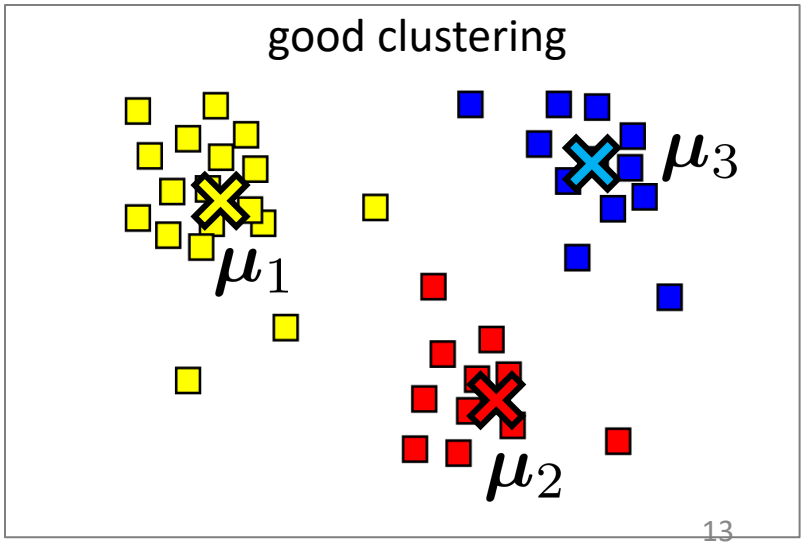
Useful for computer vision systems

Delong et al 2012 😬

K-means clustering

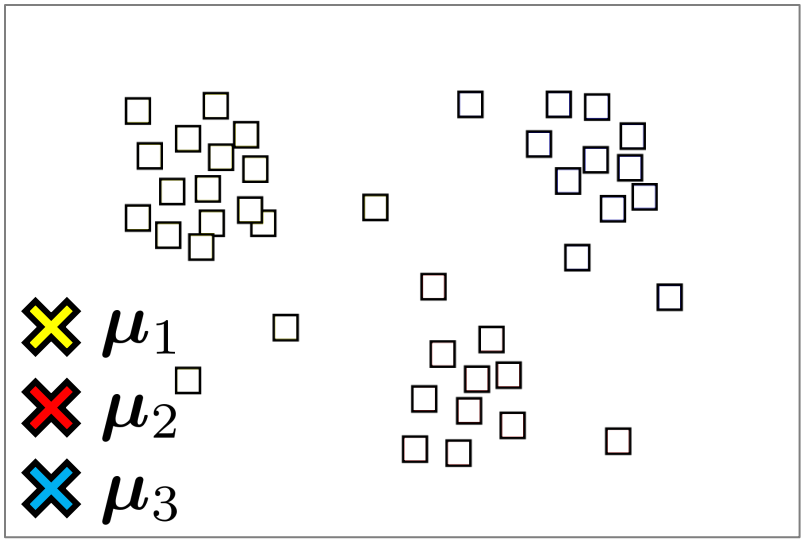
- Goal: partition $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ into *K* groups *i.e.* associate each $\mathbf{x}_i$ with a label from 1..*K*
- Idea: Introduce 'centroids' $\{\boldsymbol{\mu}_1, \ldots \boldsymbol{\mu}_K\}$, then adjust both the centroids and the cluster assignments until each $\mathbf{x}_i$ is close to its assigned centroid.

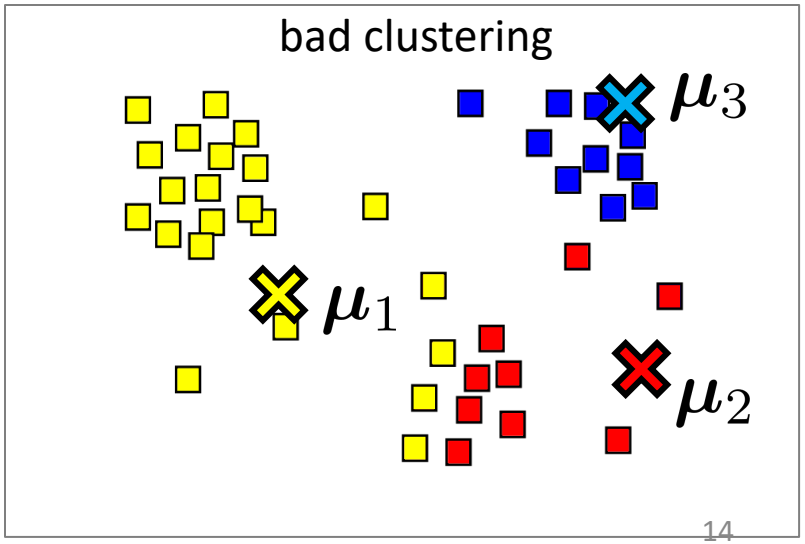




K-means clustering

- Goal: partition $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ into *K* groups *i.e.* associate each $\mathbf{x}_i$ with a label from 1..*K*
- Idea: Introduce 'centroids' $\{\boldsymbol{\mu}_1, \ldots \boldsymbol{\mu}_K\}$, then adjust both the centroids and the cluster assignments until each $\mathbf{x}_i$ is close to its assigned centroid.





K-means clustering objective

- The *K*-means algorithm tries to solve the following constrained minimization problem:

$$\min_{\mathbf{r},\boldsymbol{\mu}} \sum_{i=1}^{N}\sum_{k=1}^{K} r_{ik} \left\|\mathbf{x}_i - \boldsymbol{\mu}_k\right\|^2$$

quantity called the "distortion"

$$\text{s.t.} \sum_{k=1}^{K} r_{ik} = 1$$

$$r_{ik} \in \{0, 1\}$$

- Quadratic in $\boldsymbol{\mu}_k$, but no closed-form solution for both $\boldsymbol{\mu}_k, r_{ik}$ and no gradient w.r.t. r_{ik} (discrete).

K-means algorithm ('learning')

- Idea 1: given fixed $\{\boldsymbol{\mu}_1, \ldots \boldsymbol{\mu}_K\}$, can we at least say what the optimal r_{ik} assignments should be?
- Yes! Simply associate point *i* with its nearest centroid

$$r_{ik} = \begin{cases} 1 & \text{if } k = \arg\min_j \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2 \\ 0 & \text{otherwise.} \end{cases}$$

K-means algorithm ('learning')

- Idea 2: if we assume r_{ik} are fixed, can say what the optimal centroid placements $\{\boldsymbol{\mu}_1, \ldots \boldsymbol{\mu}_K\}$ are?
- Yes! The K-means objective is quadratic in $\boldsymbol{\mu}_k$ so we can directly solve by setting gradient to 0.

$$\nabla_{\boldsymbol{\mu}_k} \left[\sum_{i=1}^{N} \sum_{j=1}^{K} r_{ij} \left\| \mathbf{x}_i - \boldsymbol{\mu}_j \right\|^2 \right] = 2 \sum_{i=1}^{N} r_{ik} (\boldsymbol{\mu}_k - \mathbf{x}_i) = 0$$

$$\Rightarrow \boldsymbol{\mu}_k = \frac{\sum_{i=1}^{N} r_{ik} \mathbf{x}_i}{\sum_{i=1}^{N} r_{ik}}$$

The *mean* of all $\mathbf{x}_i$ for which $r_{ik} = 1$

(There are K of them, hence "K-means")

K-means algorithm ('learning')

Hard assignment step ("E step"):

$$r_{ik} = \begin{cases} 1 & \text{if } k = \arg\min_j \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2 \\ 0 & \text{otherwise.} \end{cases}$$

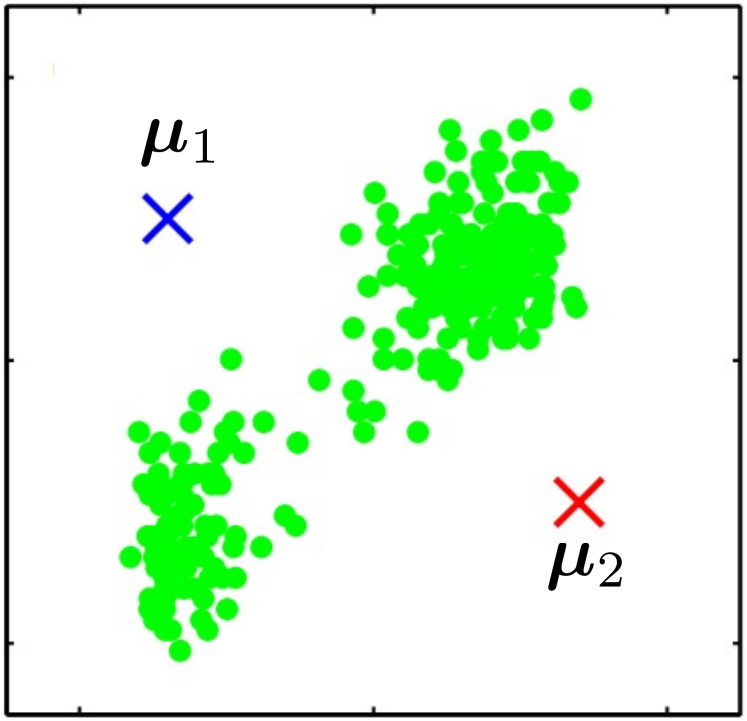
Mean update step ("M step"):

$$\boldsymbol{\mu}_k = \frac{\sum_{i=1}^{N} r_{ik}\mathbf{x}_i}{\sum_{i=1}^{N} r_{ik}}$$

Stop when nothing changes.

Called a "coordinate descent" algorithm because it "descends" with respect to only subset of parameters (coordinates) in any given step.

K-means example



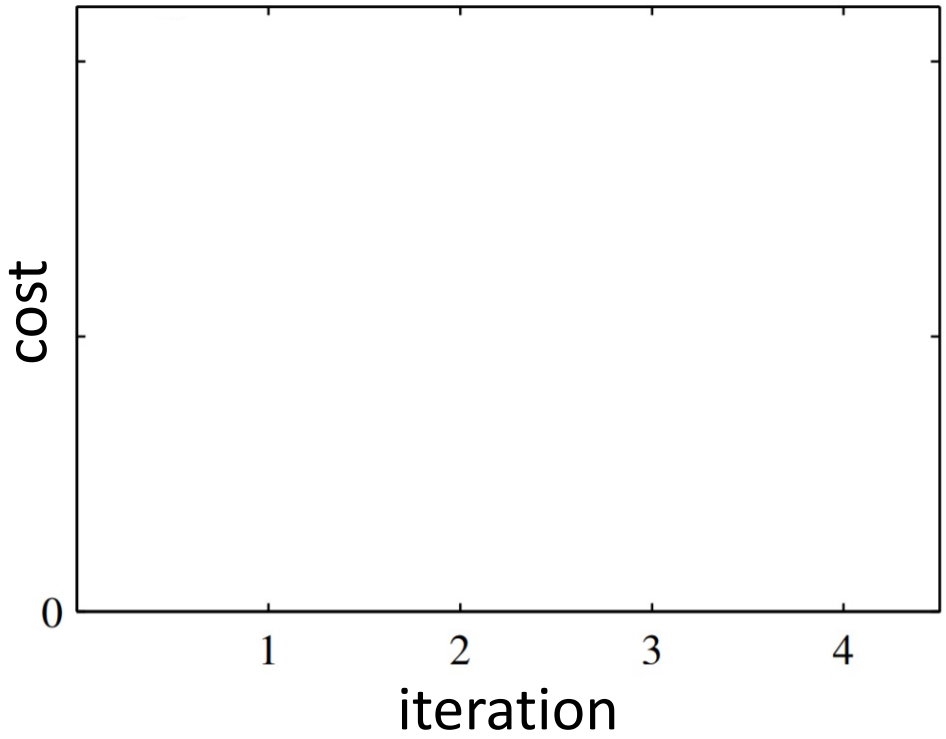
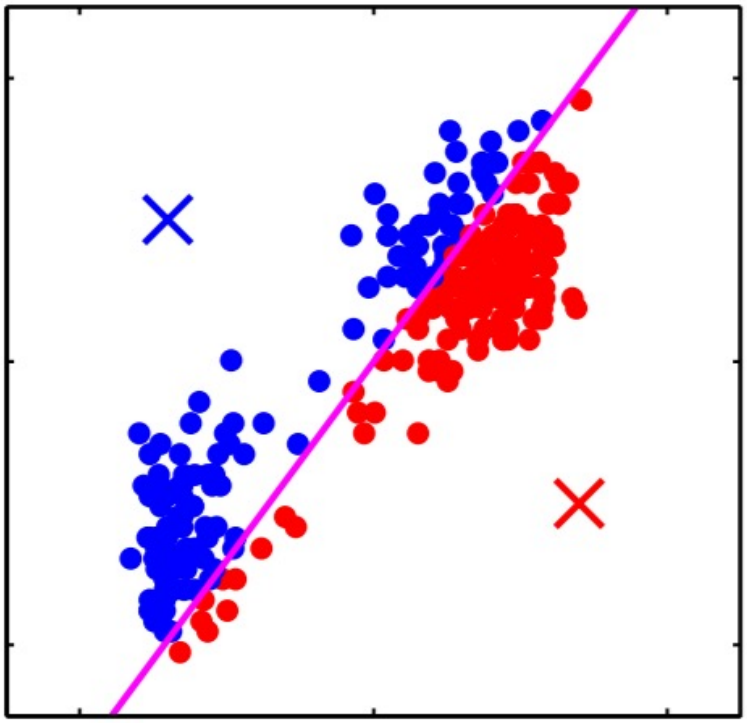
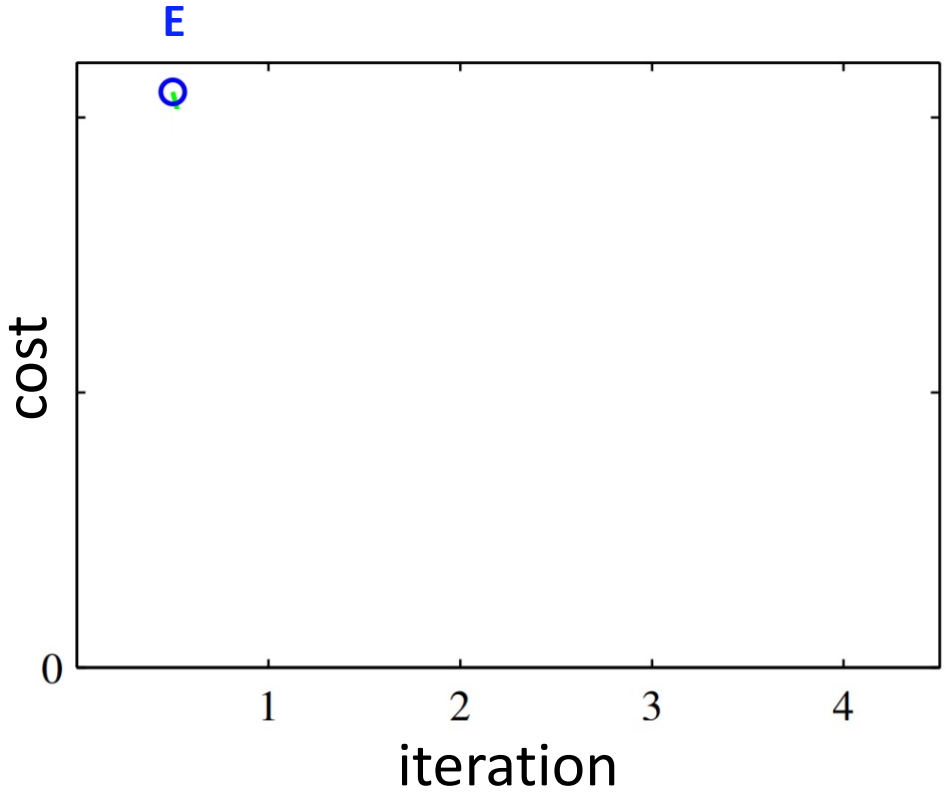


Image credit: Christopher M. Bishop

K-means example

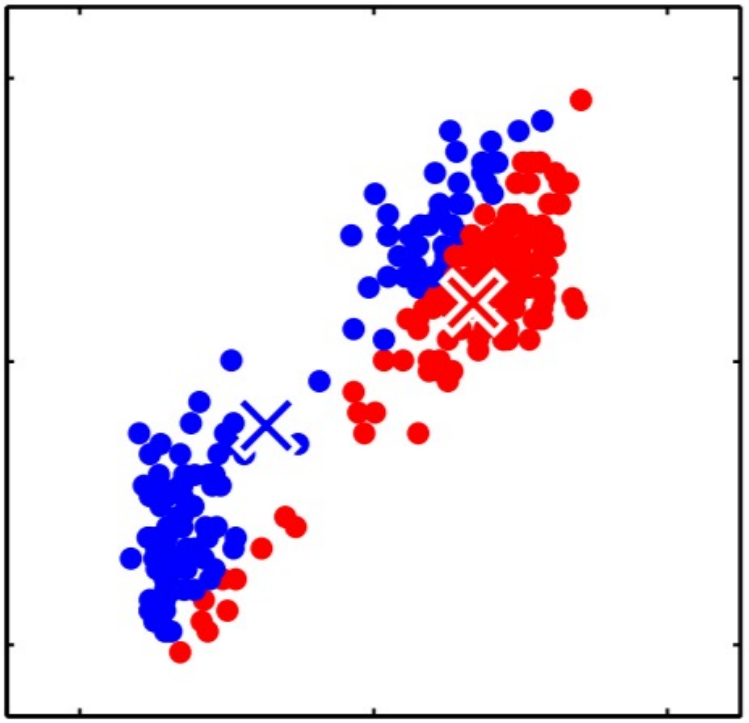


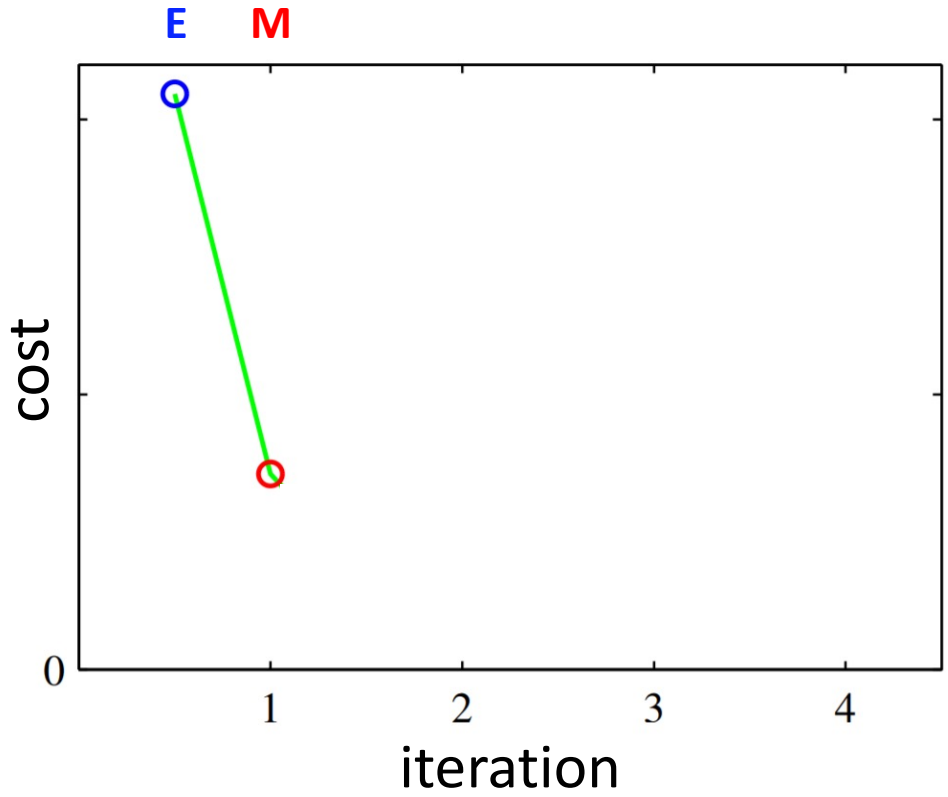


minimize over $\mathbf{r}$

Image credit: Christopher M. Bishop

K-means example

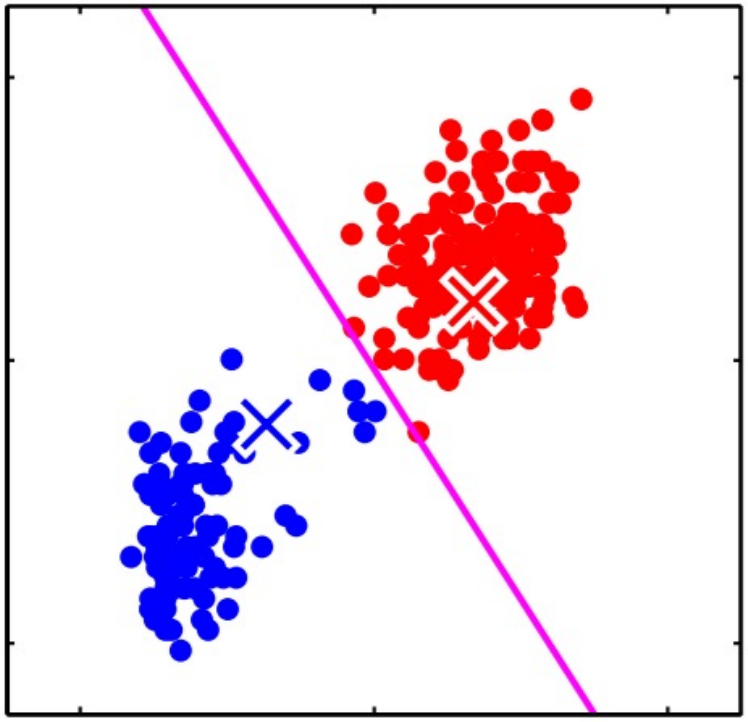


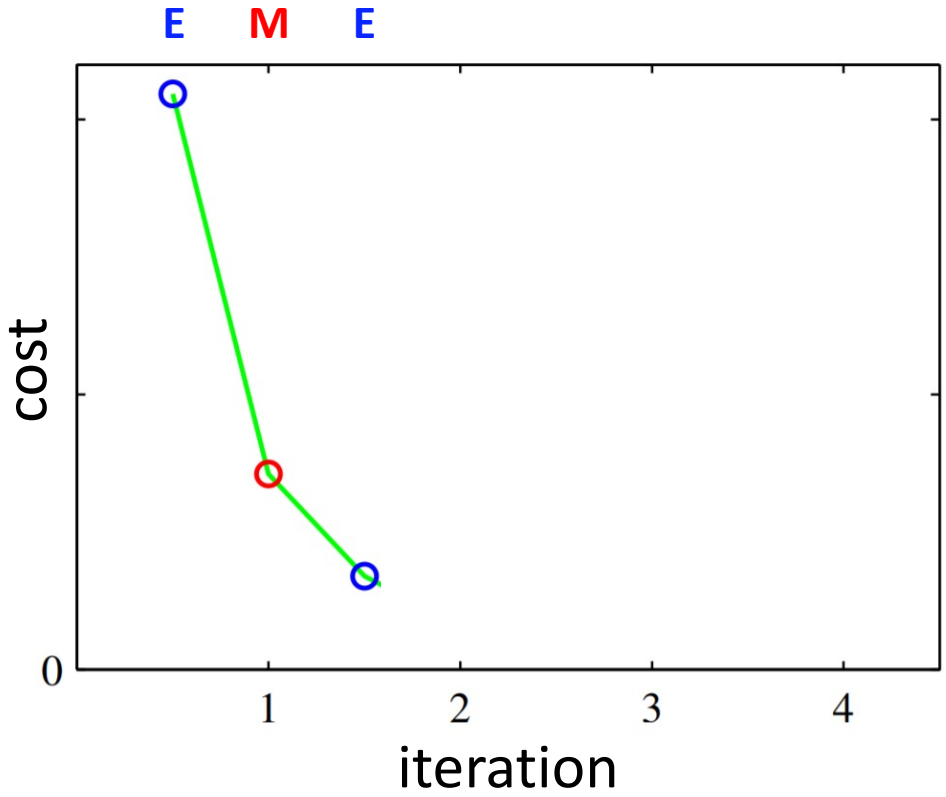


$$\text{minimize over } \{\boldsymbol{\mu}_1, \ldots \boldsymbol{\mu}_K\}$$

Image credit: Christopher M. Bishop

K-means example

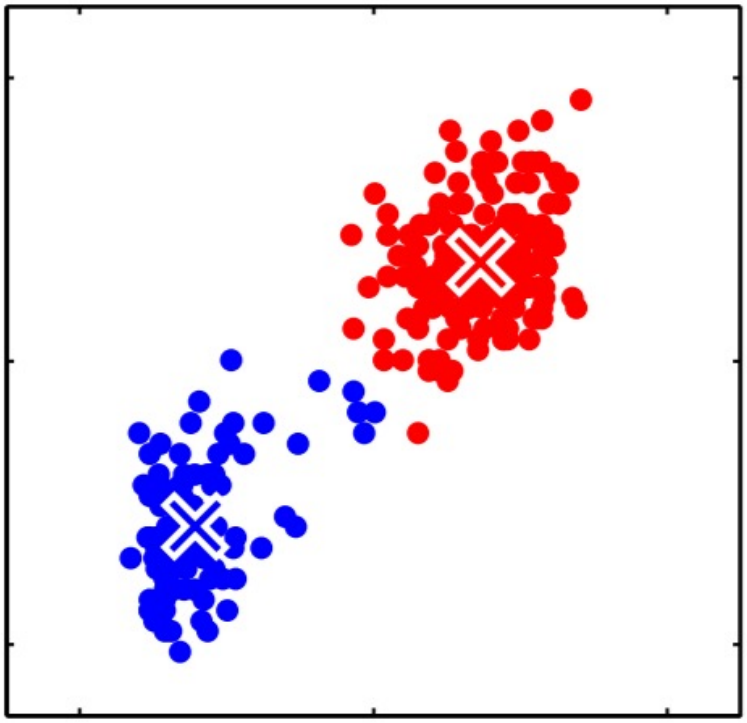


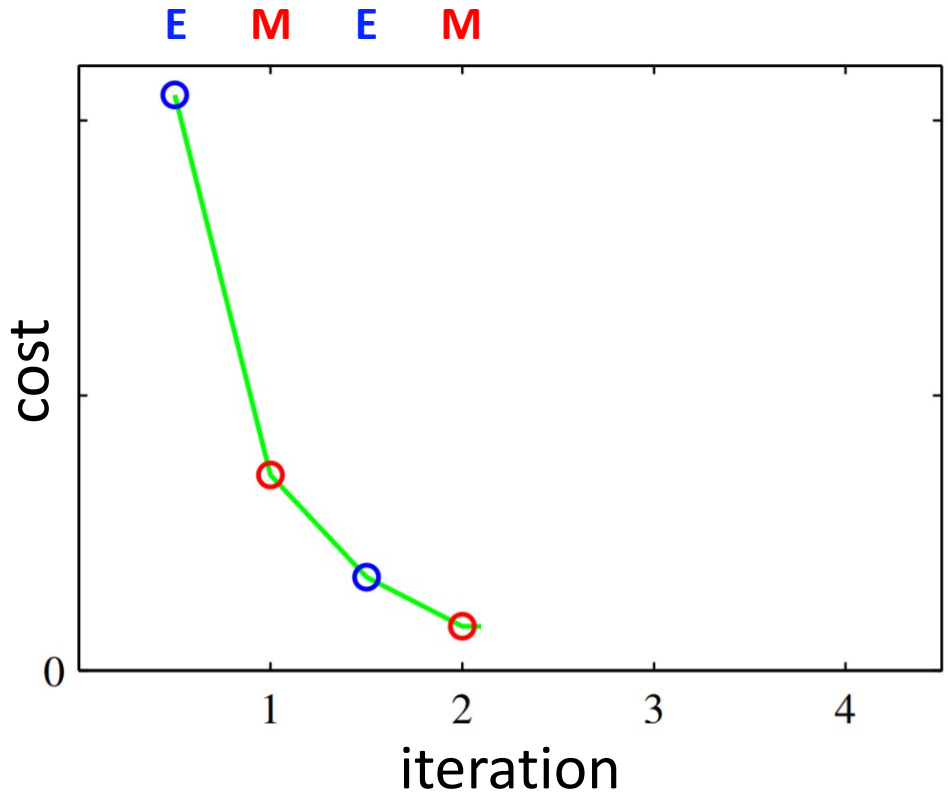


minimize over $\mathbf{r}$

Image credit: Christopher M. Bishop

K-means example

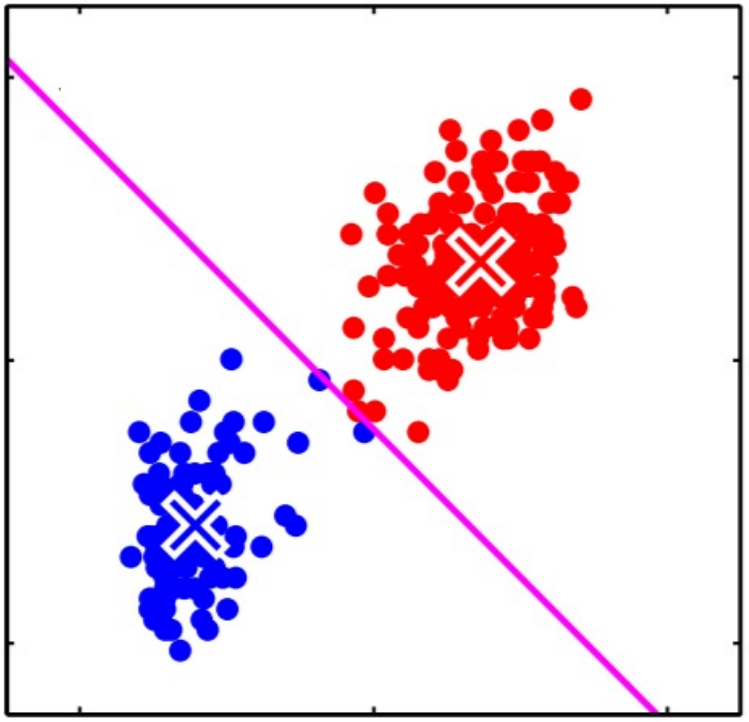


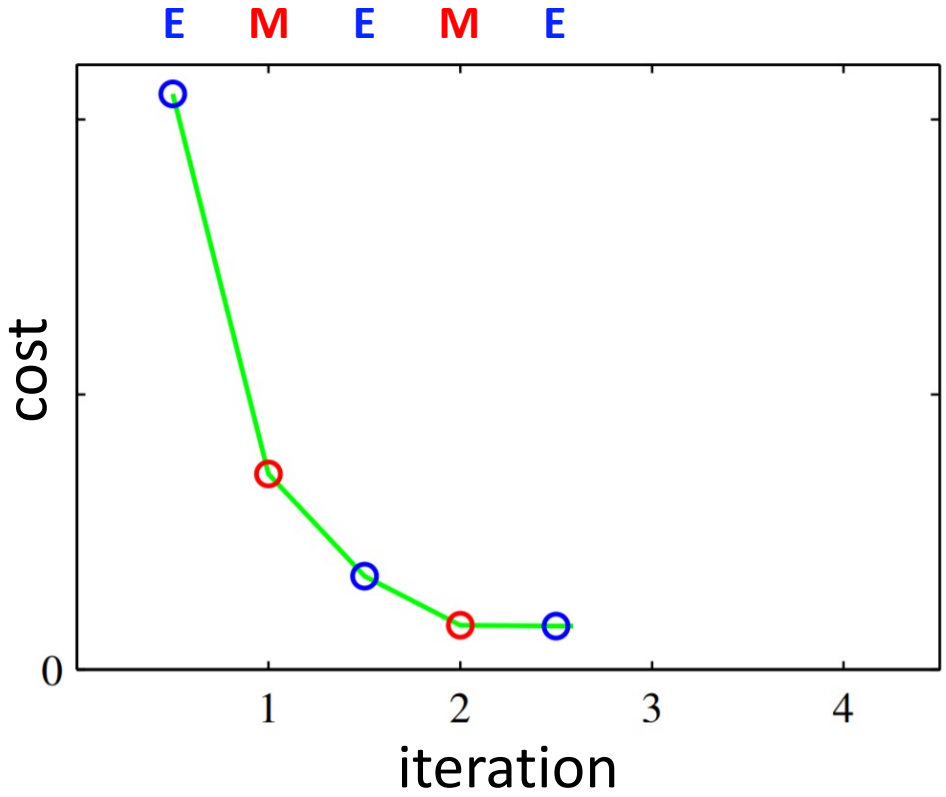


$$\text{minimize over } \{\boldsymbol{\mu}_1, \ldots \boldsymbol{\mu}_K\}$$

Image credit: Christopher M. Bishop

K-means example

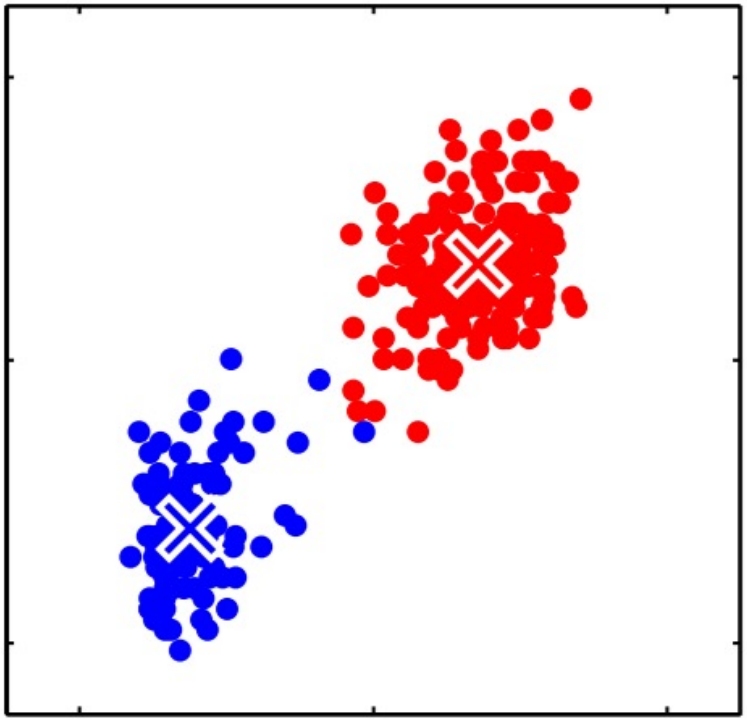


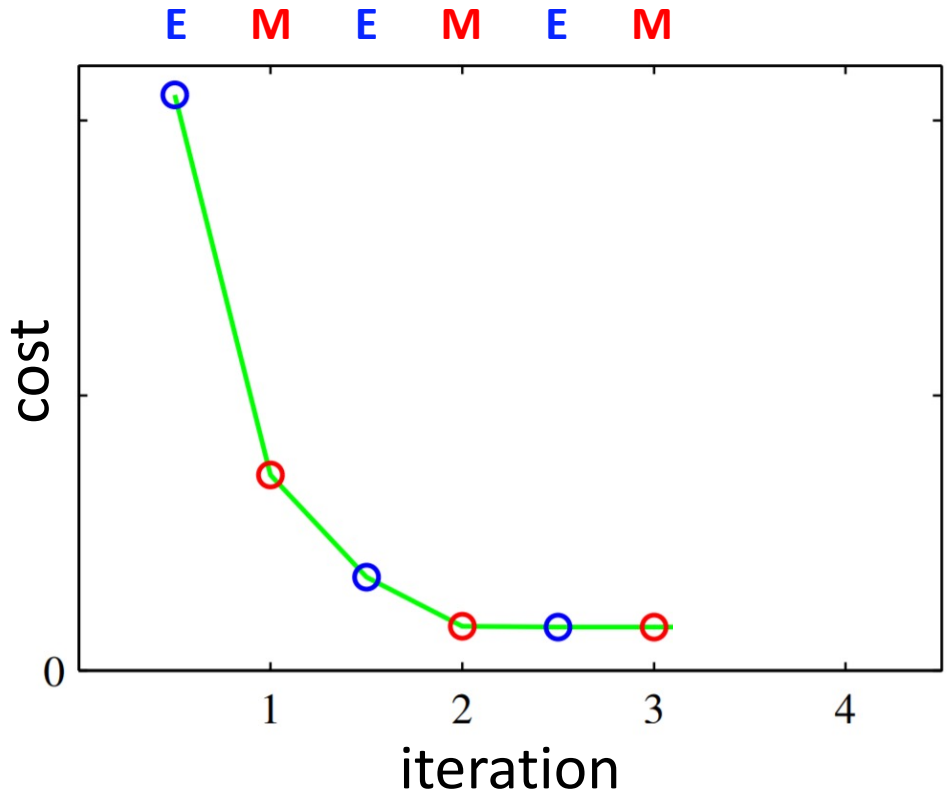


minimize over $\mathbf{r}$

Image credit: Christopher M. Bishop

K-means example

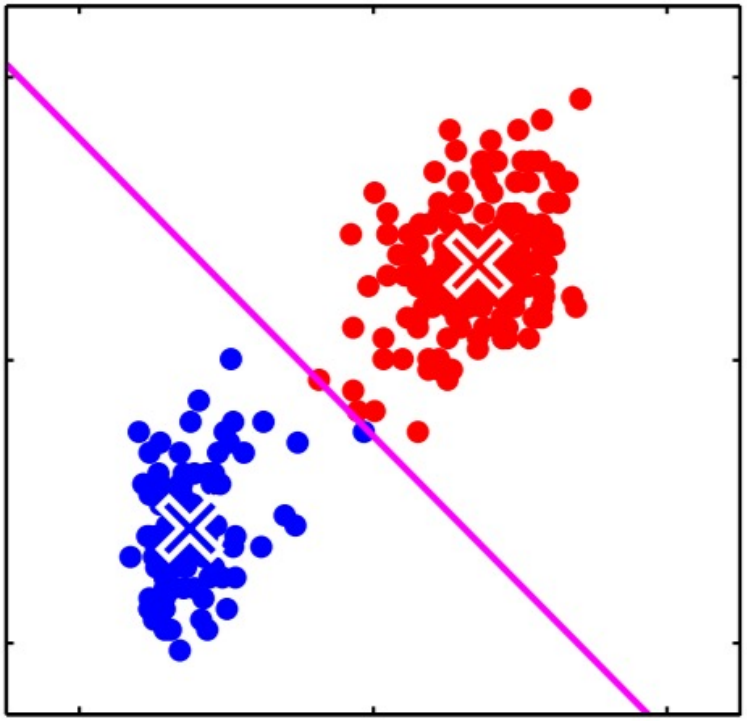


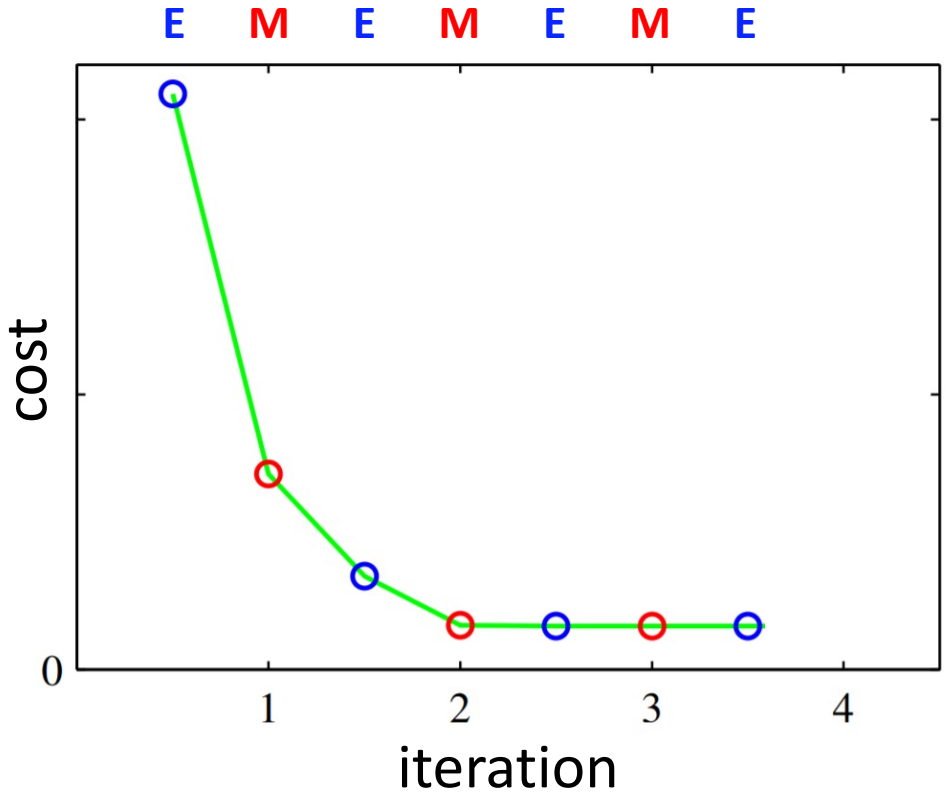


$$\text{minimize over } \{\boldsymbol{\mu}_1, \ldots \boldsymbol{\mu}_K\}$$

Image credit: Christopher M. Bishop

K-means example

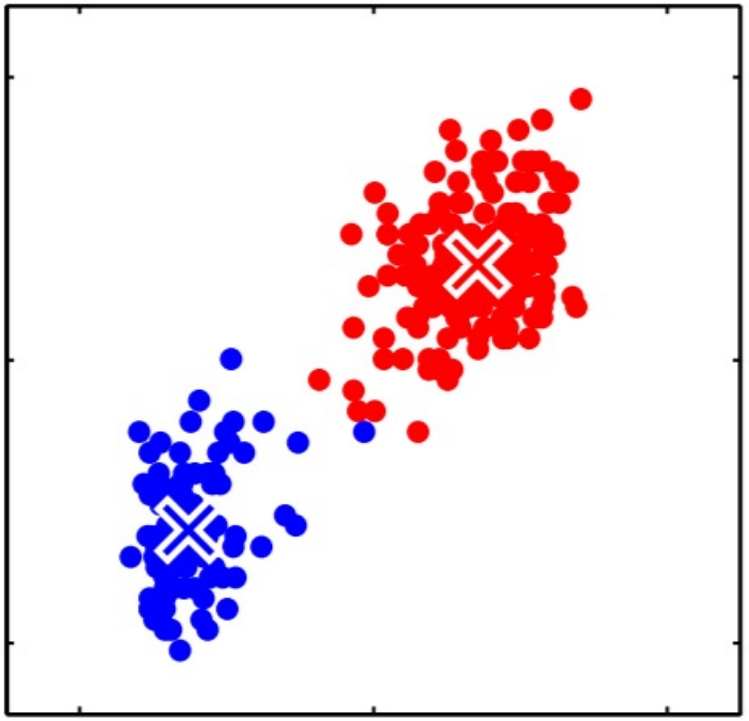


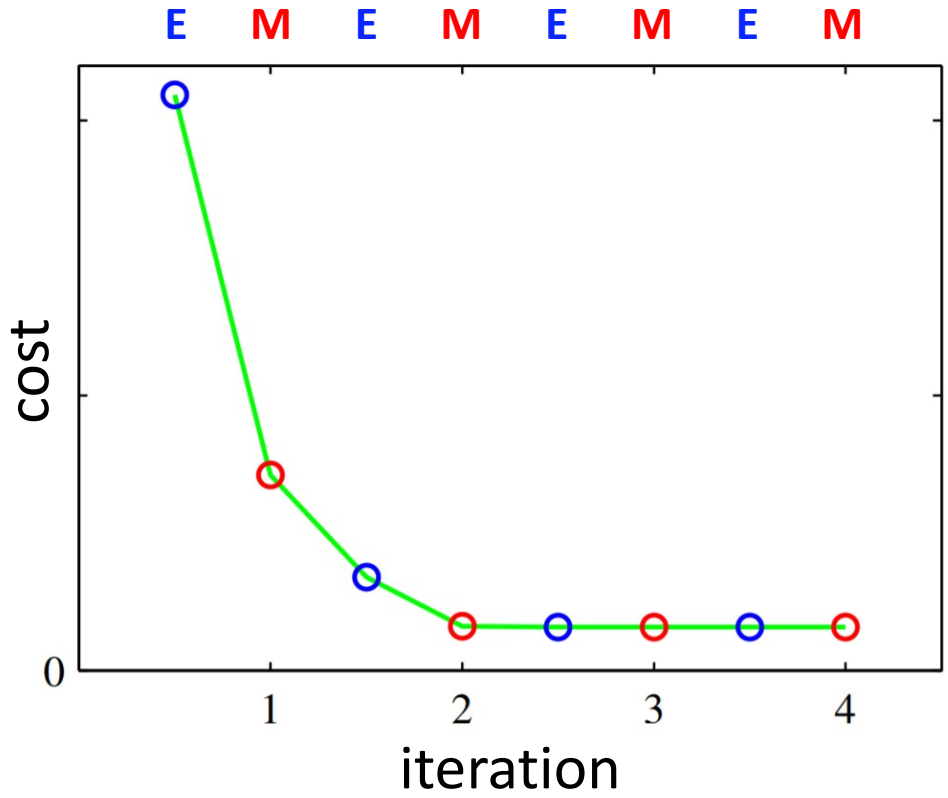


minimize over $\mathbf{r}$

Image credit: Christopher M. Bishop

K-means example





$\text{minimize over } \{\boldsymbol{\mu}_1, \ldots \boldsymbol{\mu}_K\}$

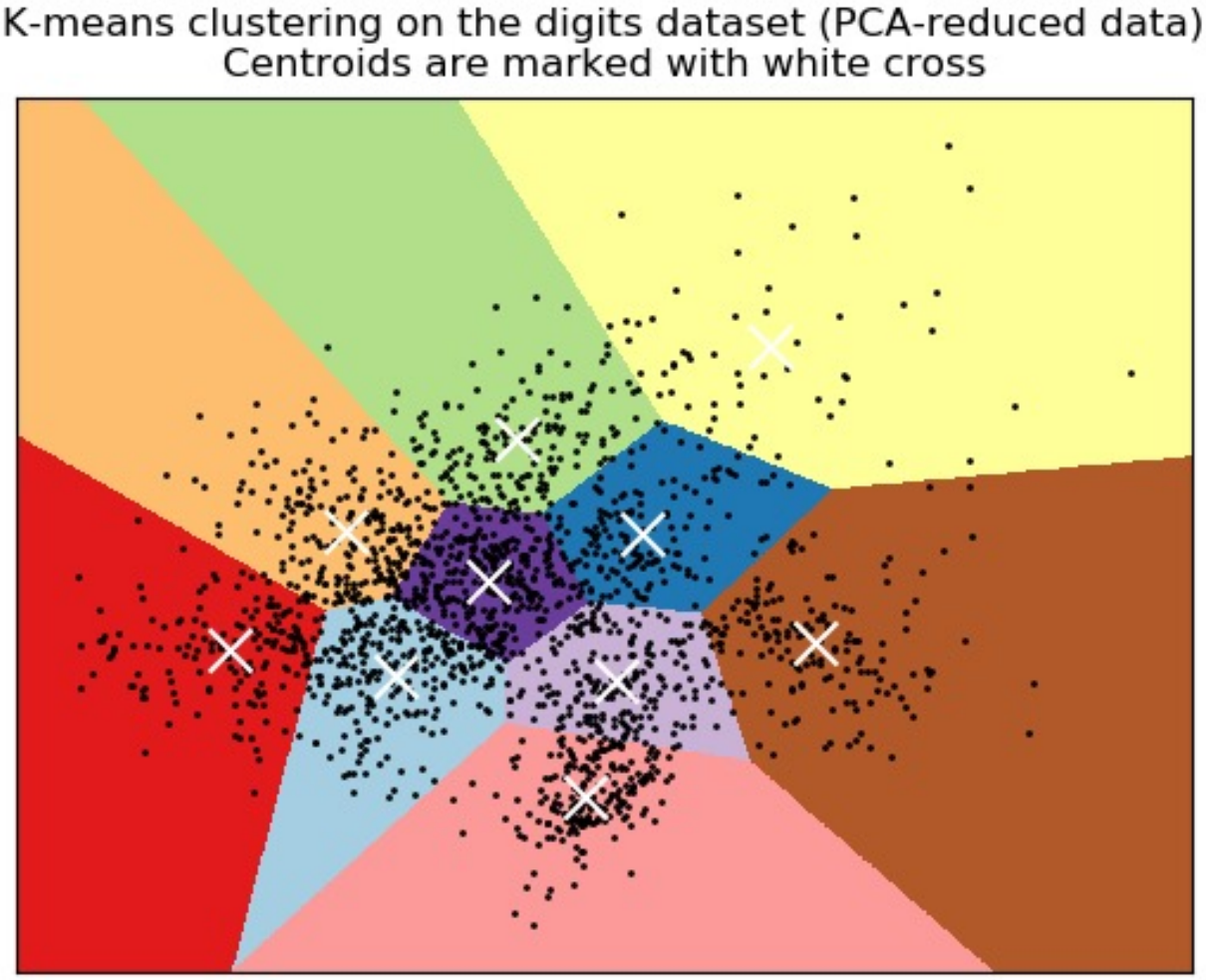
DONE!

Image credit: Christopher M. Bishop

K-means initialization

- Coordinate descent can get stuck in a local minimum. No approximation guarantee!
- Choice of initial centroids $\{\boldsymbol{\mu}_1, \ldots \boldsymbol{\mu}_K\}$ has big impact on chance of success (good clustering)
- **Random:** set initial $\boldsymbol{\mu}_k$ to be a random data point selected from $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$
- **K-means++**: set initial $\boldsymbol{\mu}_k$ to be far away from $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_{k-1}\}$ with high probability (spread)
  - The default for scikit-learn's KMeans class

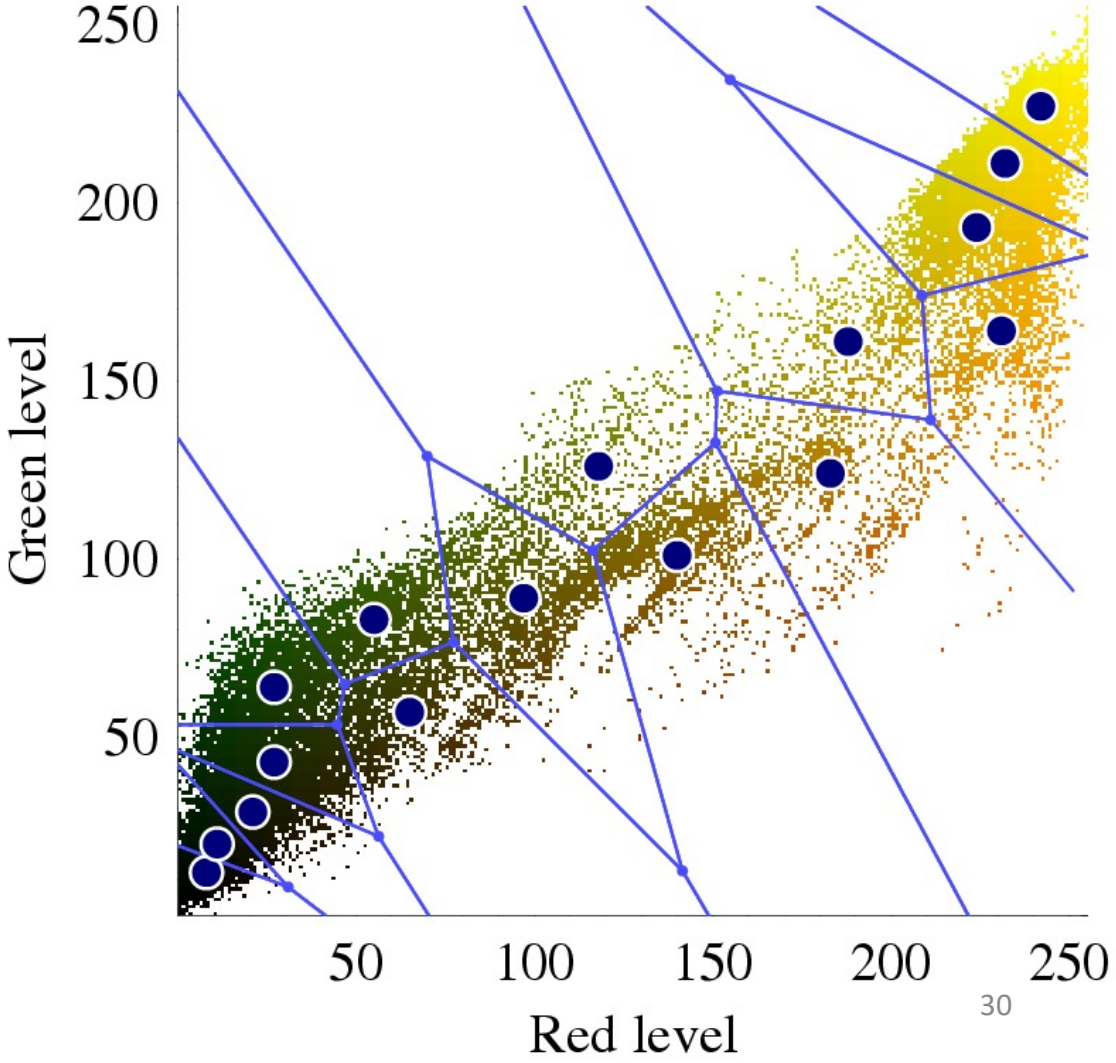
K-means and Voronoi cells



Source: scikit-learn.org

K-means and Voronoi cells

K=16, 4 bits per pixel



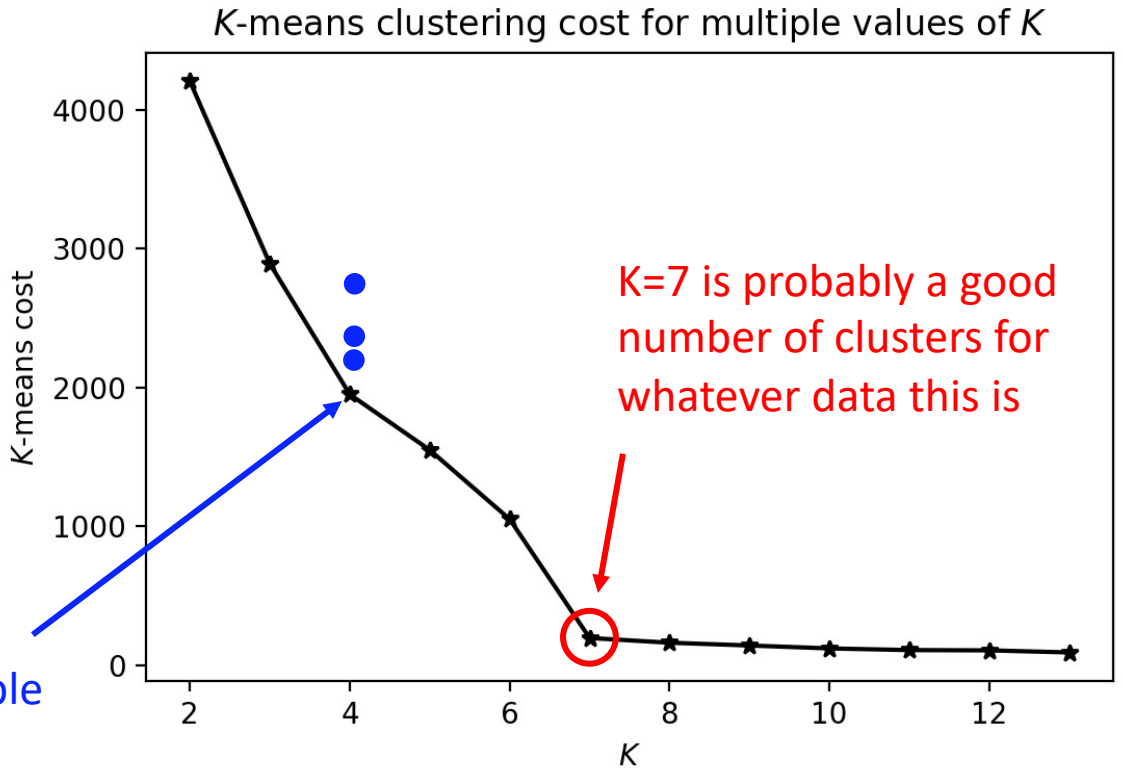
Source: Wikipedia

K-means limitations

- Must guess *K* up front, or try many *K* and select one via a model selection criterion.
- Assumes clusters of equal size; biased against finding mix of small and large clusters
  - Can be extended to "weighted *K*-means"
- Assumes clusters are isotropic; does not model covariance or elongated structure
  - Can be extended to "elliptical *K*-means"
- Assumes $\|\mathbf{x} - \mathbf{y}\|^2$ is a useful distance measure
  - Can be extended to "kernel *K*-means"

Choosing *K*

- Can try multiple values of *K*, plot the 'cost' of each clustering, and look for an 'elbow' inflection point:



Each of these should be the best from multiple random "seeds" (trials)

K-means limitations

elliptical *K*-means
(Sung & Poggio 1995)

***weighted* elliptical *K*-means**

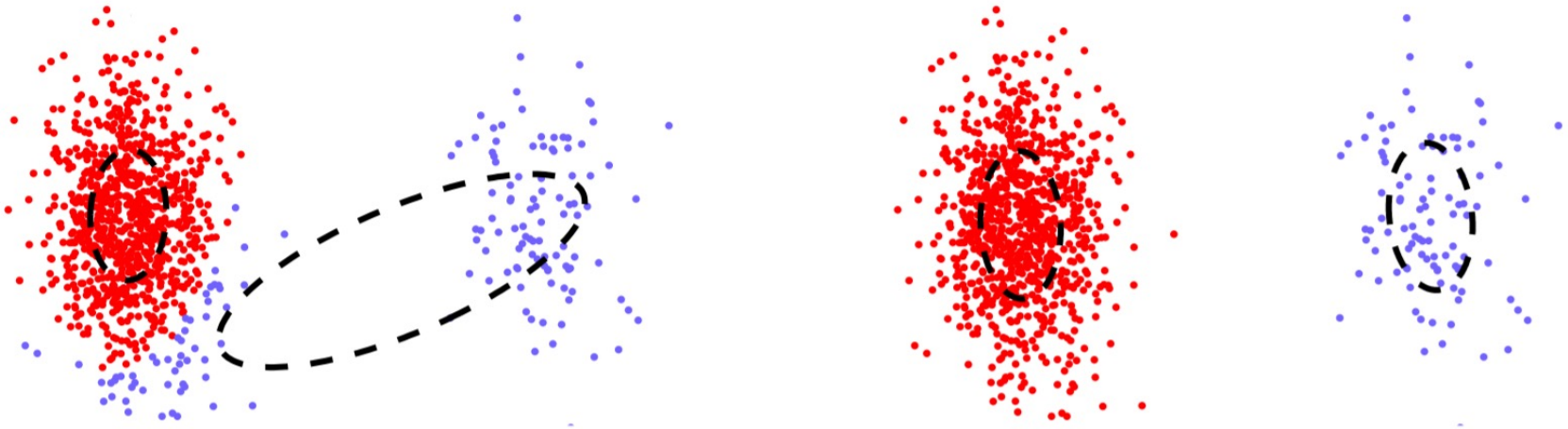
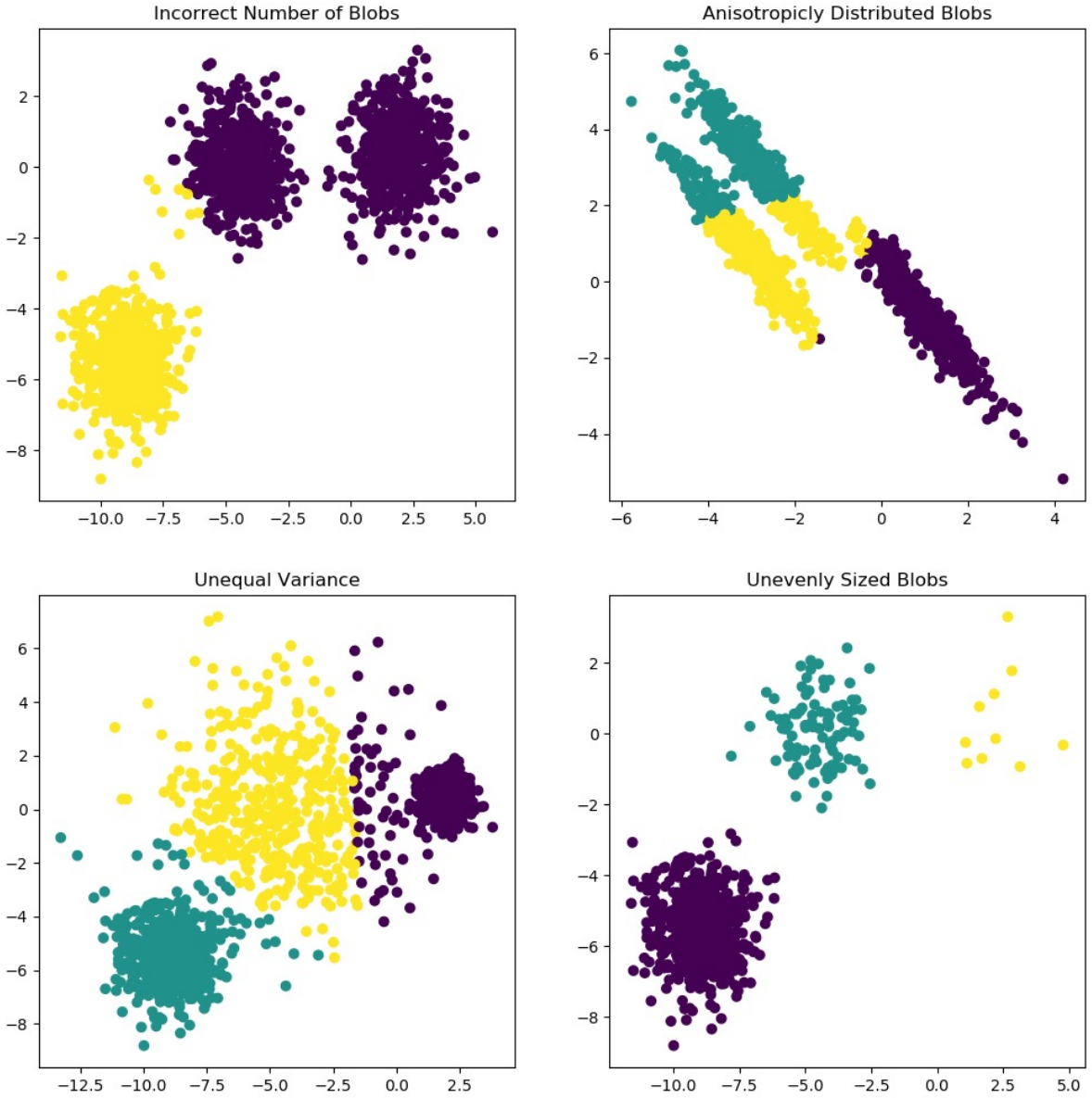


Fig. 9 Mixture of two Gaussians where most data points were generated from the first component. Standard K-means prefers equal cluster sizes, whereas weighted K-means has no such bias.

Delong et al 2012

Basic K-means limitations



Source: scikit-learn.org

Integer *K*-means formulation

- Can define $\mathbf{r} = \begin{bmatrix} r_1 & \cdots & r_N \end{bmatrix}^T$ to be a vector of integer 'labels' and use them as index into the mean $\boldsymbol{\mu}_{r_i}$ currently associated with $\mathbf{x}_i$

$$\min_{\mathbf{r},\boldsymbol{\mu}} \sum_{i=1}^{N} \|\mathbf{x}_i - \boldsymbol{\mu}_{r_i}\|^2$$

$$\text{s.t. } r_i \in \{1, \ldots, K\}$$

- Closer to how real implementations work
  - The earlier $r_{ik} \in \{0, 1\}$ formulation helps understand relation of K-means to EM-algorithm in next section!

class `sklearn.cluster.` **KMeans** (*n_clusters=8, init='k-means++', n_init=10, max_iter=300, tol=0.0001, precompute_distances='auto', verbose=0, random_state=None, copy_x=True, n_jobs=None, algorithm='auto'*)

[source]

K-Means clustering

Read more in the User Guide.

Parameters:

n_clusters : *int, optional, default: 8*

The number of clusters to form as well as the number of centroids to generate.

init : *{'k-means++', 'random' or an ndarray}*

Method for initialization, defaults to 'k-means++':

'k-means++' : selects initial cluster centers for k-mean clustering in a smart way to speed up convergence. See section Notes in k_init for more details.

'random': choose k observations (rows) at random from data for the initial centroids.

If an ndarray is passed, it should be of shape (n_clusters, n_features) and gives the initial centers.

n_init : *int, default: 10* ← number of randomized trials

Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia.

max_iter : *int, default: 300*

Maximum number of iterations of the k-means algorithm for a single run.

Example from scikit-learn docs

Examples

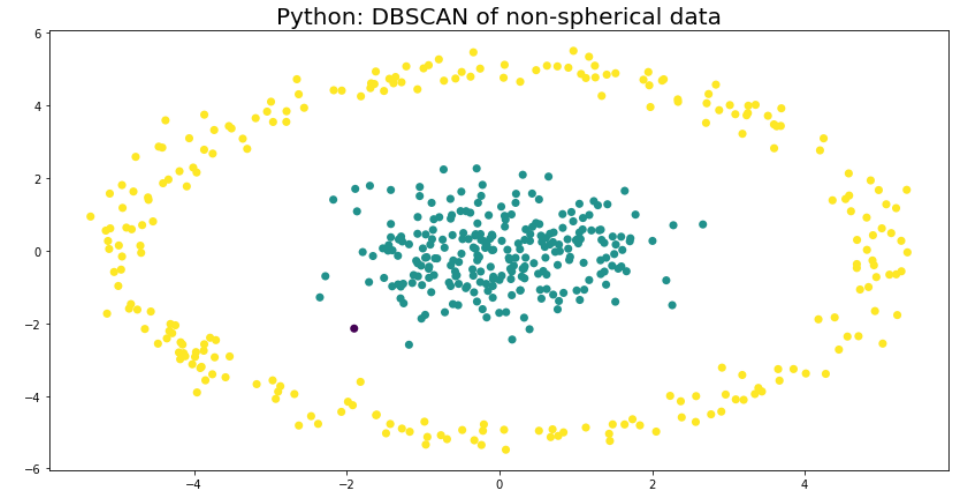
```
>>> from sklearn.cluster import KMeans
>>> import numpy as np
>>> X = np.array([[1, 2], [1, 4], [1, 0],
...               [10, 2], [10, 4], [10, 0]])
>>> kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
>>> kmeans.labels_
array([1, 1, 1, 0, 0, 0], dtype=int32)
>>> kmeans.predict([[0, 0], [12, 3]])
array([1, 0], dtype=int32)
>>> kmeans.cluster_centers_
array([[10.,  2.],
       [ 1.,  2.]])
```

integer formulation

associate new points to the centroids

DBSCAN: Popular alternative to *K*-means

- Based on "growing" clusters from the most dense regions
- Handles arbitrary cluster topology, including non-spherical clusters
- Can handle outliers better than *K*-means
- Specify distance threshold and sample threshold rather than specify *K*, which may (or may not) be more intuitive



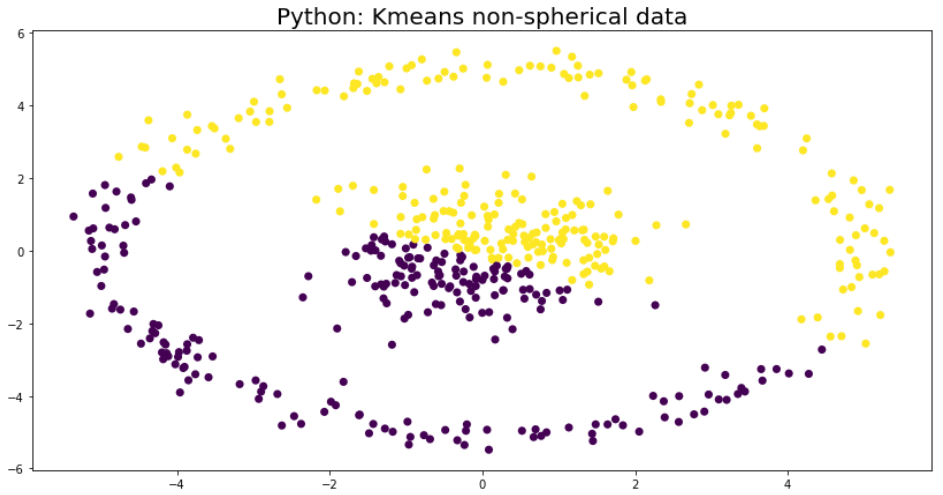
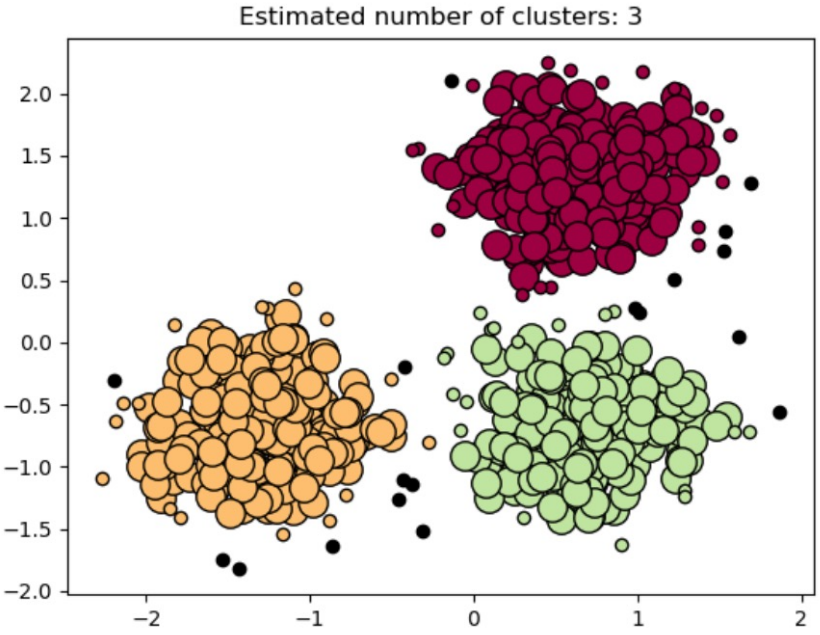


Image credit: Fisseha Berhane https://datascience-enthusiast.com/Python/DBSCAN_Kmeans.html

DBSCAN: Popular alternative to *K*-means

sklearn.cluster.DBSCAN

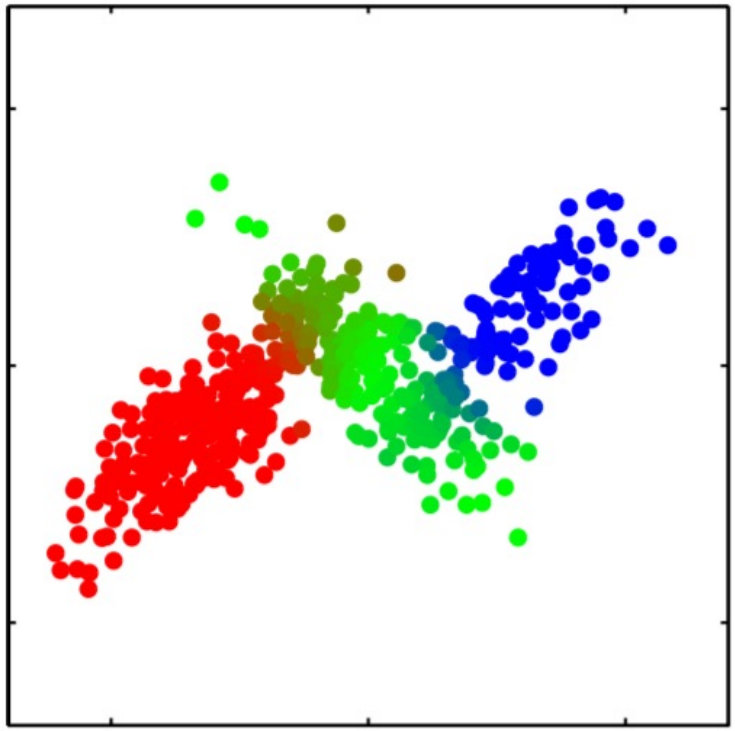
class `sklearn.cluster.` **DBSCAN** (*eps=0.5, min_samples=5, metric='euclidean', metric_params=None, algorithm='auto', leaf_size=30, p=None, n_jobs=None*)



Out:

```
Estimated number of clusters: 3
Estimated number of noise points: 18
Homogeneity: 0.953
```

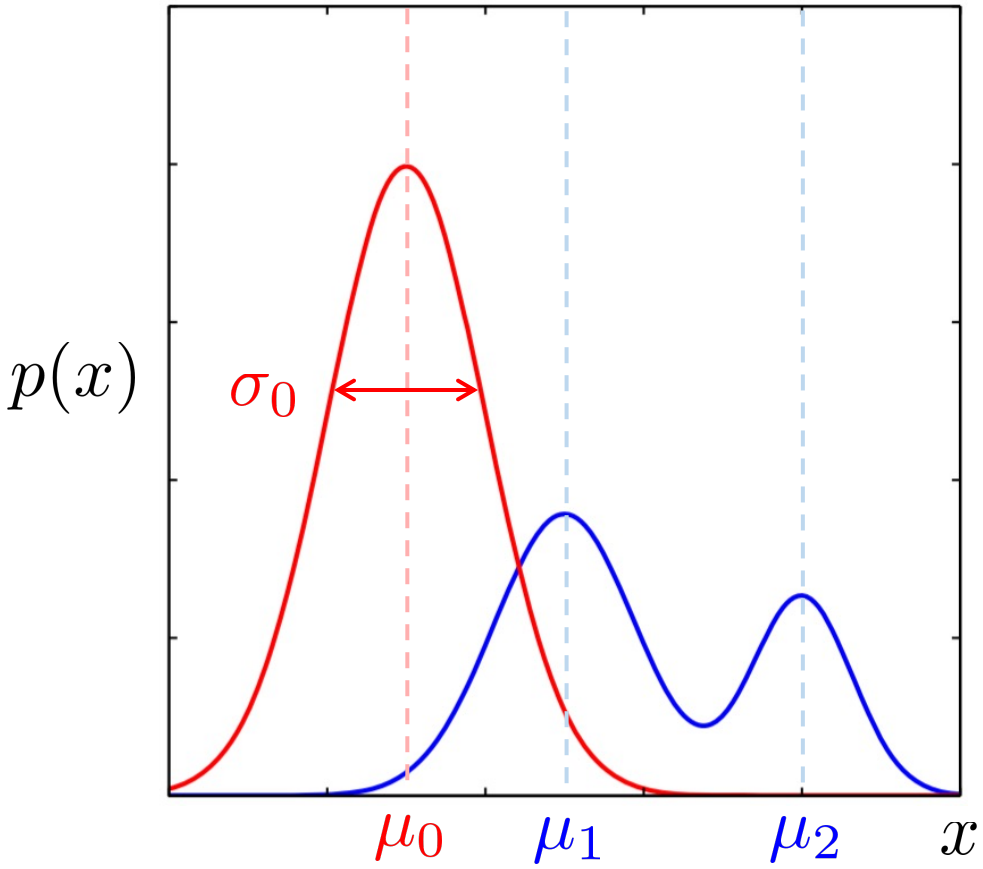
Gaussian Mixture Models



Gaussian Mixture Model (GMM)

$$p(x) = \mathcal{N}(x \mid \mu_0, \sigma_0^2)$$

$$p(x) = 0.7\mathcal{N}(x \mid \mu_1, \sigma_1^2) + 0.3\mathcal{N}(x \mid \mu_2, \sigma_2^2)$$

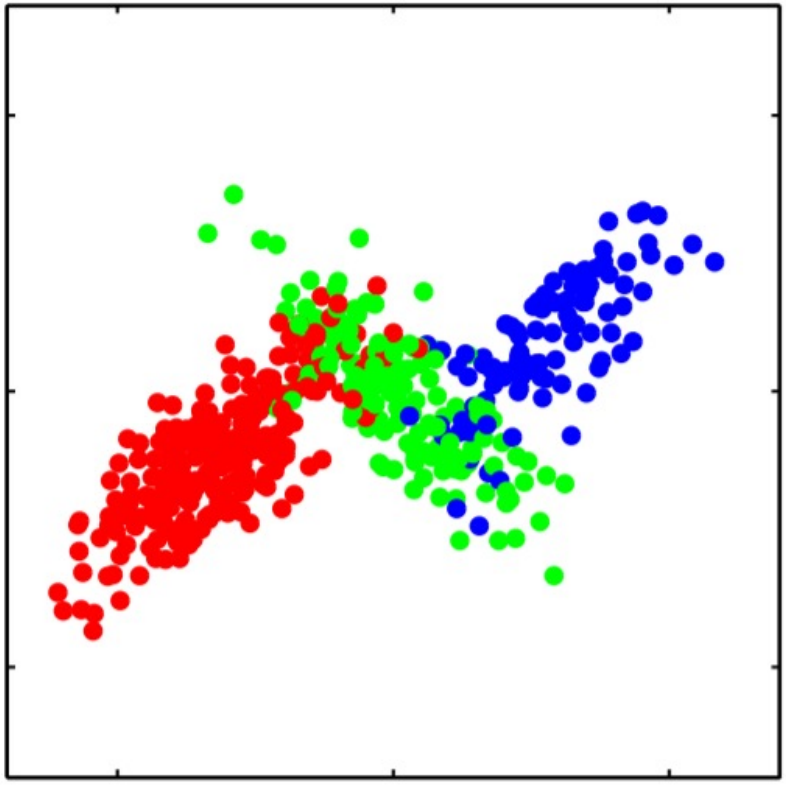


This particular distribution is *univariate* and *bimodal* (two "modes"), with a *major* mode and a *minor* mode.

Note that its expected value (its mean) is not equal to either of the modes.

Gaussian Mixture Model (GMM)

- A Gaussian Mixture distribution is a weighted superposition of individual Gaussian distributions



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

$$\text{s.t.} \sum_{k=1}^{K} \pi_k = 1$$

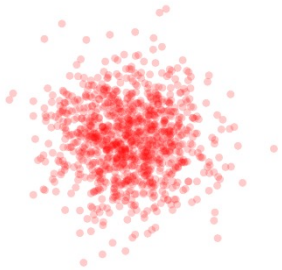
$$\pi_k \geq 0$$

Covariance matrix

- A multivariate normal distribution $\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$ is specified by a *mean vector* and a symmetric positive semidefinite *covariance matrix*

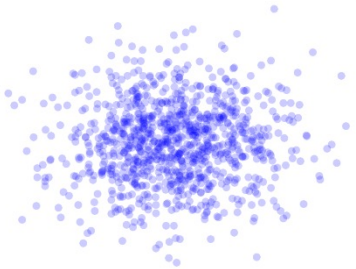
$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix} = \sigma^2 \mathbf{I}$$

isotropic (spherical)



$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix}$$

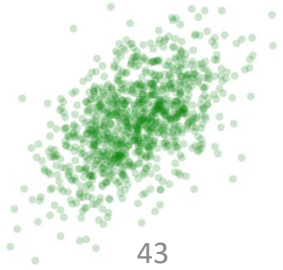
dimensions uncorrelated



$$\boldsymbol{\Sigma} = \begin{bmatrix} \mathrm{Cov}[x_1, x_1] & \mathrm{Cov}[x_1, x_2] \\ \mathrm{Cov}[x_2, x_1] & \mathrm{Cov}[x_2, x_2] \end{bmatrix}$$

(symmetric)

dimensions can be correlated



Multivariate normal

- Probability density of a sample $\mathbf{x} \in \mathbb{R}^D$ from a *D*-dimensional multivariate normal distribution is:

$$\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{Z} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}$$

$$Z = (2\pi)^{\frac{D}{2}} \det(\boldsymbol{\Sigma})^{\frac{1}{2}}$$

- When *D*=1, reduces to standard formula

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

numpy.random.multivariate_normal

numpy.random.**multivariate_normal**(*mean*, *cov*[, *size*, *check_valid*, *tol*])

Draw random samples from a multivariate normal distribution.

The multivariate normal, multinormal or Gaussian distribution is a generalization of the one-dimensional normal distribution to higher dimensions. Such a distribution is specified by its mean and covariance matrix. These parameters are analogous to the mean (average or "center") and variance (standard deviation, or "width," squared) of the one-dimensional normal distribution.

Parameters:

mean : *1-D array_like, of length N*

Mean of the N-dimensional distribution.

cov : *2-D array_like, of shape (N, N)*

Covariance matrix of the distribution. It must be symmetric and positive-semidefinite for proper sampling.

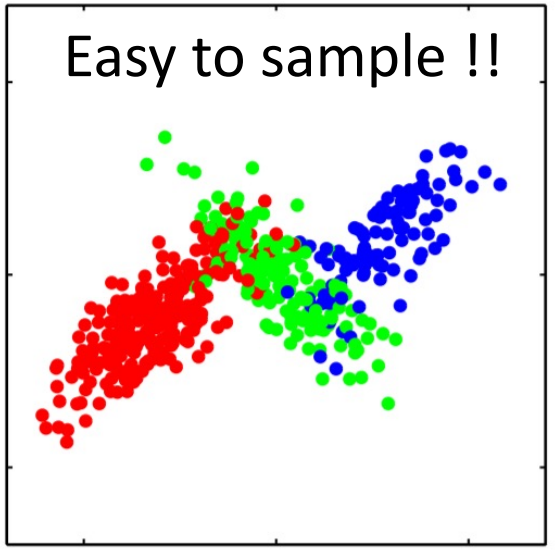
size : *int or tuple of ints, optional*

Given a shape of, for example, `(m,n,k)`, `m*n*k` samples are generated, and packed in an *m*-by-*n*-by-*k* arrangement. Because each sample is *N*-dimensional, the output shape is `(m,n,k,N)`. If no shape is specified, a single (*N*-D) sample is returned.

Sampling from a GMM

1. Sample integer $k \in \{1, \ldots, K\}$ with probability π_k of being selected
2. Sample a data point according to

$$\mathbf{x} \sim \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$



number of mixture components to select from

number of independent samples to draw

Sampling a batch of component indices in Numpy according to probabilities:

```
>>> np.random.choice(5, 3, p=[0.1, 0, 0.3, 0.6, 0])
array([3, 3, 0])
```

probability of each integer

Sampling from a GMM

- Can capture complex high-dimensional structure

Richardson & Weiss 2018 https://arxiv.org/abs/1805.12462

Sampling from a GMM

- Can sample only missing data according to the GMM distribution

Richardson & Weiss 2018 https://arxiv.org/abs/1805.12462

Fitting a GMM to data

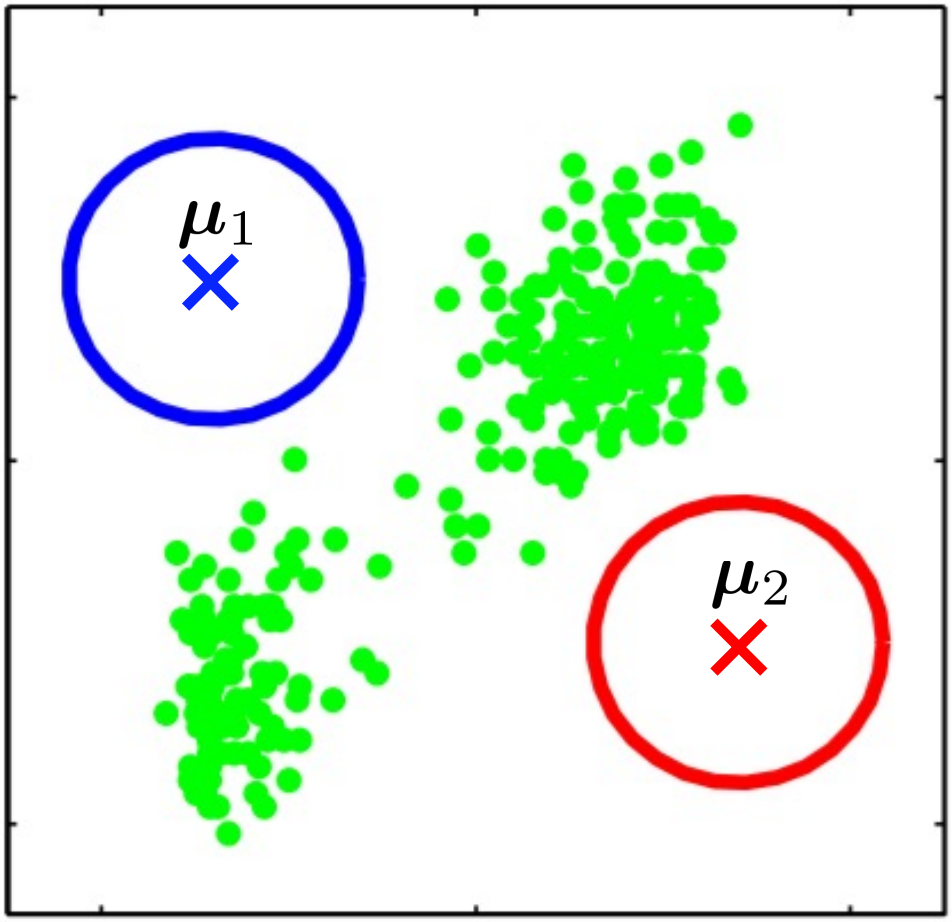
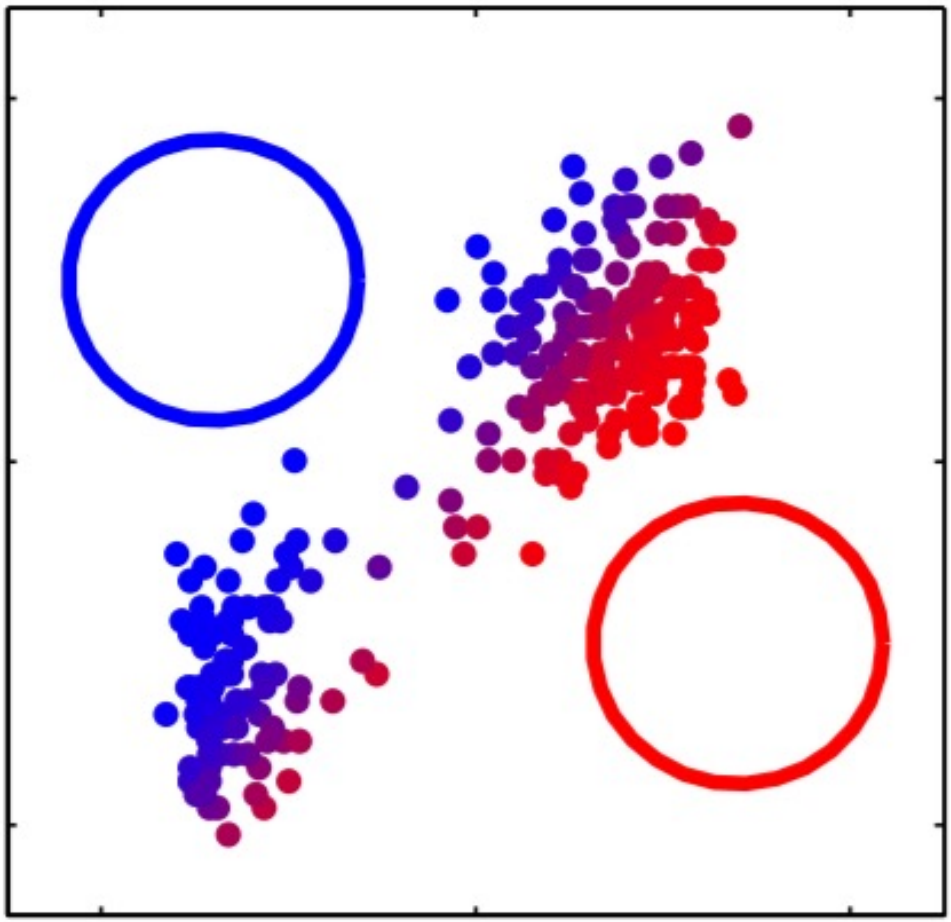


Image credit: Christopher M. Bishop

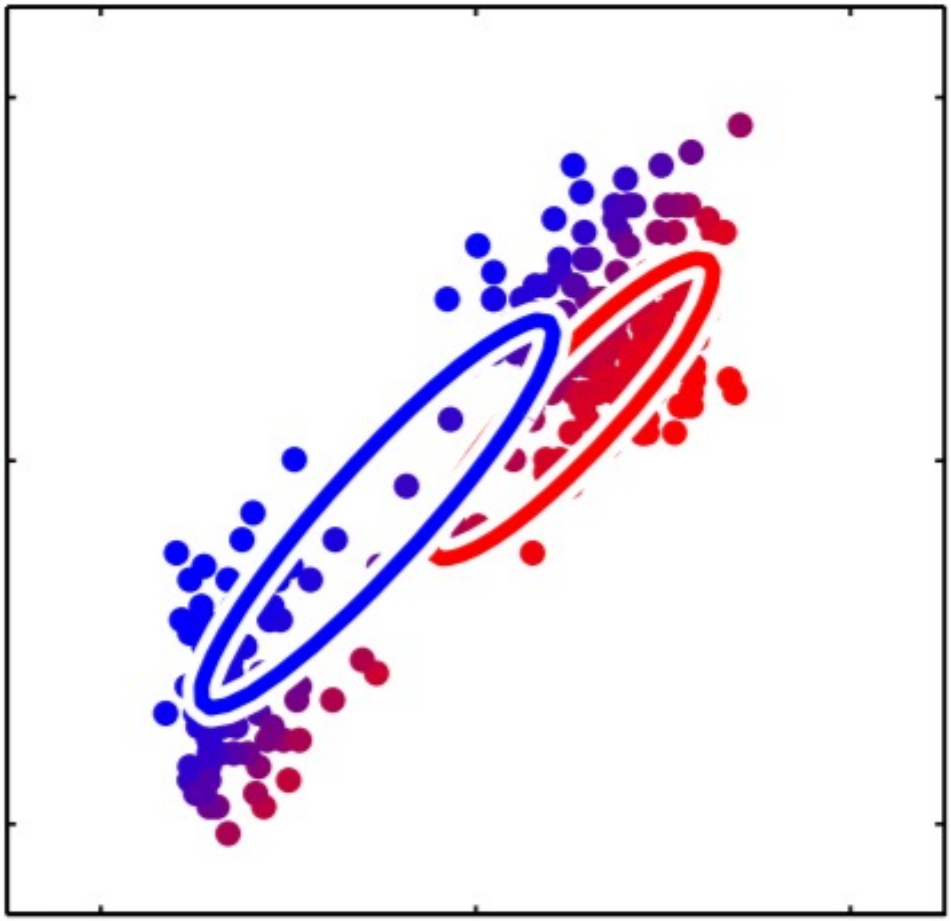
Fitting a GMM to data



initialization

Image credit: Christopher M. Bishop

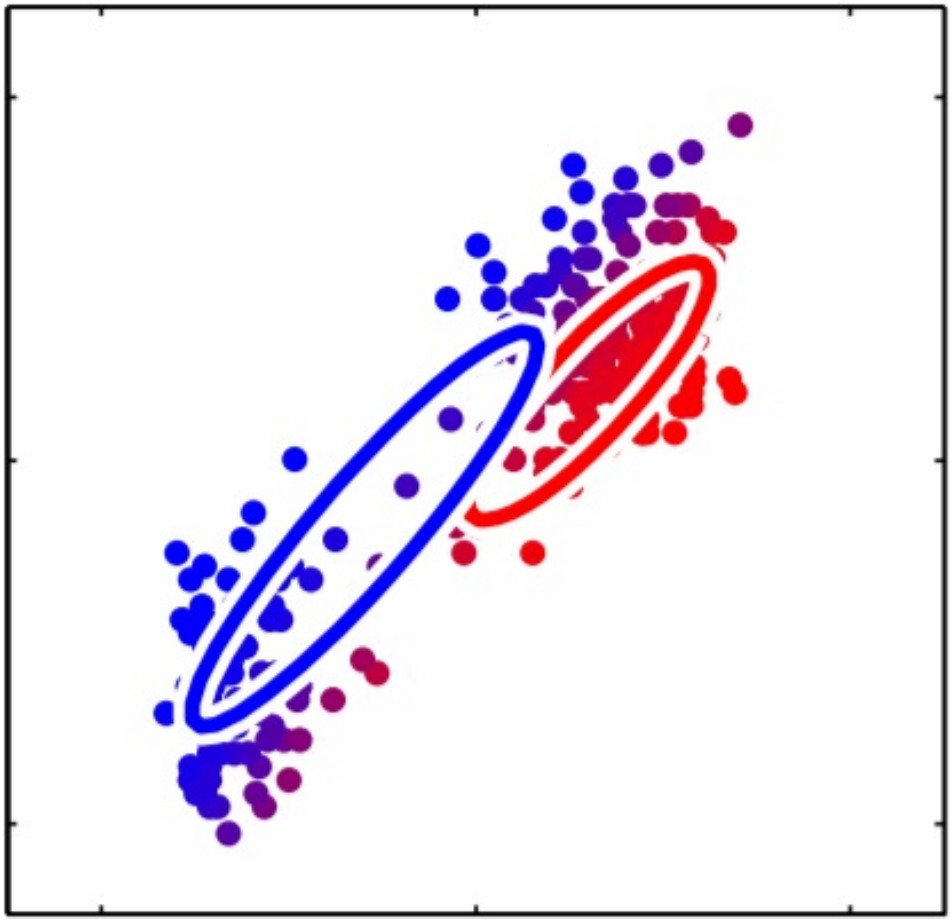
Fitting a GMM to data



1 iteration

Image credit: Christopher M. Bishop

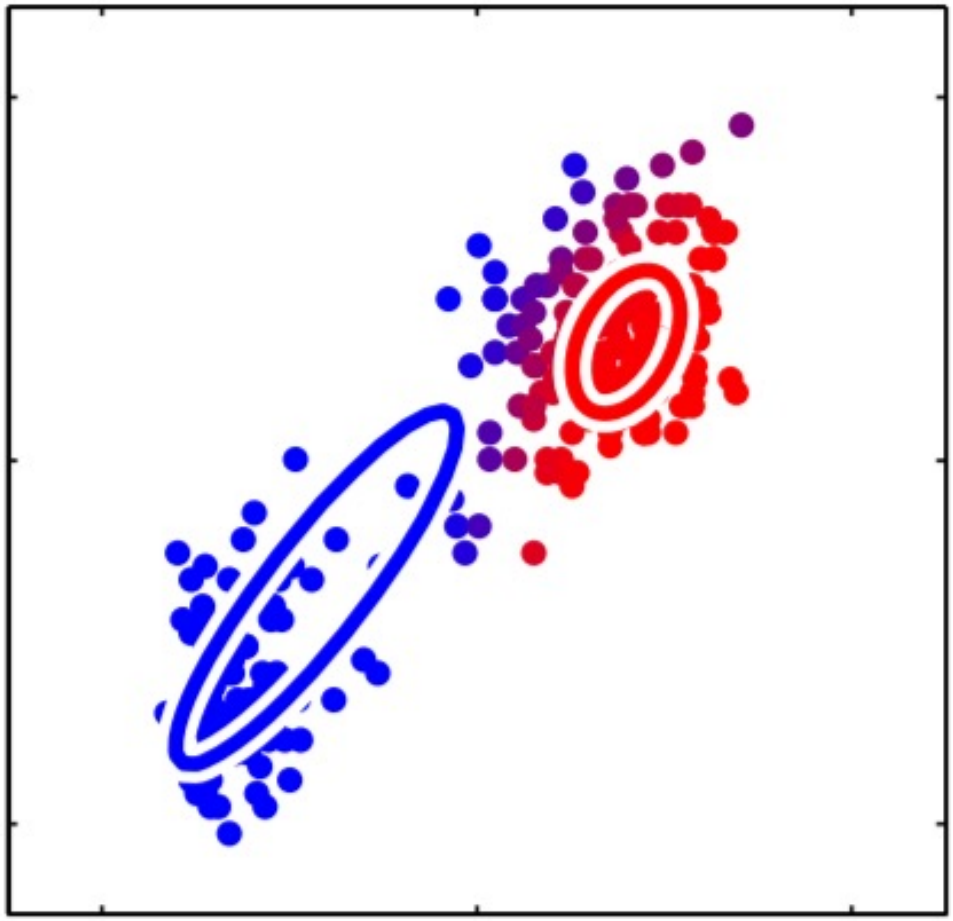
Fitting a GMM to data



2 iterations

Image credit: Christopher M. Bishop

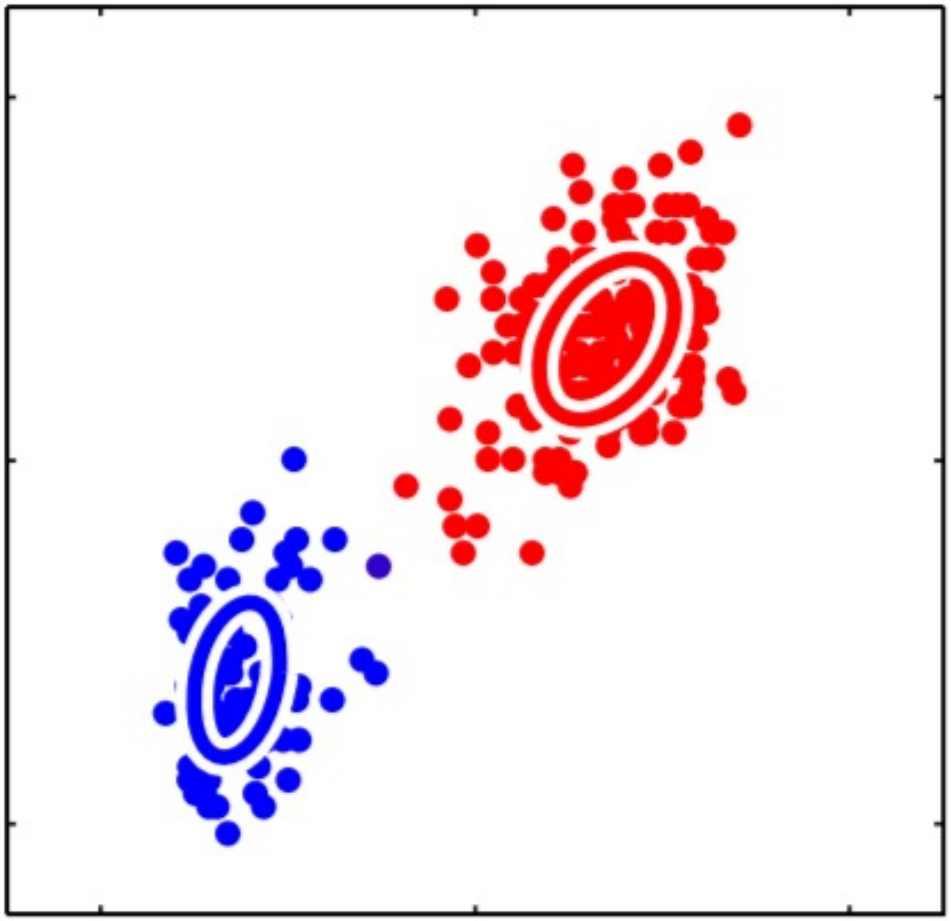
Fitting a GMM to data



5 iterations

Image credit: Christopher M. Bishop

Fitting a GMM to data



20 iterations

Image credit: Christopher M. Bishop

And now the math…

- The next several slides will explain how the famous "EM algorithm for GMMs" is derived from the maximum likelihood principle.
  - There also exist "EM algorithm for <other mixture models>"
- You will only need to know how to derive the update rule for the "means" of a GMM, *i.e.*, when you get to this formula later on in the slides:

$$\boldsymbol{\mu}_k^{\text{new}} = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} \mathbf{x}_i$$

- Good strategy for studying is to work backwards from that formula. No need to study the other update rules (covariance matrices, weights), they're similar anyway.

Probabilistic model of GMM

- What does π_k represent? The probability that mixture component *k* was selected to generate individual sample $\mathbf{x}$
- Introduce binary random variable $\mathbf{z} \in \{0,1\}^K$ to represent this event, where $z_k = 1$ indicates that component *k* was selected and therefore must have $z_1 + \cdots + z_K = 1$
- Then $\{\pi_1, \ldots, \pi_K\}$ actually defines $p(\mathbf{z})$

$$p(z_k = 1) = \pi_k, \quad \pi_k \geq 0, \quad \sum_{k=1}^{K} \pi_k = 1$$

Probabilistic model of GMM

- Once we select $z_k = 1$ we know the conditional distribution is

$$p(\mathbf{x} \mid z_k = 1) = \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

- Distribution of entire component selection variable is

$$p(\mathbf{z}) = \prod_{k=1}^{K} \pi_k^{z_k}$$

- So conditional distribution can be written as

$$p(\mathbf{x} \mid \mathbf{z}) = \prod_{k=1}^{K} \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_k}$$

Probabilistic model of GMM

- For multiple samples $\{\mathbf{x}_1, \ldots \mathbf{x}_N\}$ we would have multiple component selection variables $\{\mathbf{z}_1, \ldots \mathbf{z}_N\}$
  - But, for now, let's focus on the probability of one sample
- Since we have closed form expressions for both $p(\mathbf{x} \mid \mathbf{z})$ and $p(\mathbf{z})$, can write $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x} \mid \mathbf{z})p(\mathbf{z})$

$$p(\mathbf{x}, \mathbf{z}) = \left(\prod_{k=1}^{K} \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_k} \right) \left(\prod_{k=1}^{K} \pi_k^{z_k} \right)$$

$$= \prod_{k=1}^{K} \left(\pi_k \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right)^{z_k}$$

$$= \pi_{k(\mathbf{z})} \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_{k(\mathbf{z})}, \boldsymbol{\Sigma}_{k(\mathbf{z})}) \text{ where } k(\mathbf{z}) \text{ is the } k \text{ such that } z_k = 1$$

Confusing notation alert

In previous slide, when we say this:

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x} \mid \mathbf{z})p(\mathbf{z})$$

It is actually shorthand for a longer expression:

$$p(\mathbf{x}, \mathbf{z} \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = p(\mathbf{x} \mid \mathbf{z}, \boldsymbol{\mu}, \boldsymbol{\Sigma})p(\mathbf{z}|\boldsymbol{\pi})$$

Depending on what one wants to emphasize, it is common to omit certain conditionals.

We did a similar thing when we said

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$ in the beginning

which is actually $p(\mathbf{x} \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$

Checkpoint: Why are we doing all this math??

- We are working towards the ability to find the maximum likelihood parameters $\boldsymbol{\pi}_{\mathrm{ML}}, \boldsymbol{\mu}_{\mathrm{ML}}, \boldsymbol{\Sigma}_{\mathrm{ML}}$ of a GMM w.r.t. a set of observations $\{\mathbf{x}_1, \ldots \mathbf{x}_N\}$
  - *i.e.* we want to "fit" the GMM to a bunch of data, giving a model that could have plausibly generated the data.
- To do that, we need an expression for the *likelihood function of the data* $p(\mathbf{x} \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$
- So far we have derived an expression for

$$p(\mathbf{x}, \mathbf{z} \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$$

- If we marginalize over **z**, do we recover GMM $p(\mathbf{x})$?

Does GMM formulation with **z** recover the correct likelihood?

- Marginalize out the "component selection" variable $\mathbf{z}$:

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z})$$

$$= \sum_{\mathbf{z}} \pi_{k(\mathbf{z})} \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_{k(\mathbf{z})}, \boldsymbol{\Sigma}_{k(\mathbf{z})})$$

An easy sum: only *K* possible **z** vectors! e.g., for *K*=3

[1,0,0]
[0,1,0]
[0,0,1]

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

Yes this is our original $p(\mathbf{x})$ for a GMM!

GMM likelihood for multiple points

- Given data set $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ the GMM likelihood function is

$$p(\mathbf{X} \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \prod_{i=1}^{N} p(\mathbf{x}_i \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$$

$$= \prod_{i=1}^{N} \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right)$$

A product of sums... when we take the the log of this, will the math still work out nicely?

Maximum likelihood for GMMs

- How do we find $\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}$ that maximize the likelihood of observed data $\mathbf{X}$?
- Take log of the data's likelihood under the GMM

$$\ln p(\mathbf{X} \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{i=1}^{N} \ln \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right)$$

- Now we need to maximize this.
  - Direct solve for $\boldsymbol{\pi}_{\mathrm{ML}}, \boldsymbol{\mu}_{\mathrm{ML}}, \boldsymbol{\Sigma}_{\mathrm{ML}}$? *No closed form solution*!
  - Gradient descent? *We could*, but can do better!
  - Instead we'll derive the ***Expectation Maximization*** **(EM)** algorithm, which is a coordinate descent (like *K*-means!)

Gradient of GMM log likelihood

- First, take gradient w.r.t. a single mean $\boldsymbol{\mu}_k$

$$\nabla_{\boldsymbol{\mu}_k}[\ \ln p(\mathbf{X} \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})\ ] = \sum_{i=1}^{N} \nabla_{\boldsymbol{\mu}_k}\left[\ \ln\left(\sum_j \pi_j \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\right)\ \right]$$

$$= \sum_{i=1}^{N} \frac{\nabla_{\boldsymbol{\mu}_k}\left[\ \sum_j \pi_j \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\ \right]}{\sum_j \pi_j \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

For fixed $\boldsymbol{\pi}, \boldsymbol{\Sigma}$ can find optimal $\boldsymbol{\mu}_k$ by setting this to 0 and solving for $\boldsymbol{\mu}_k$!

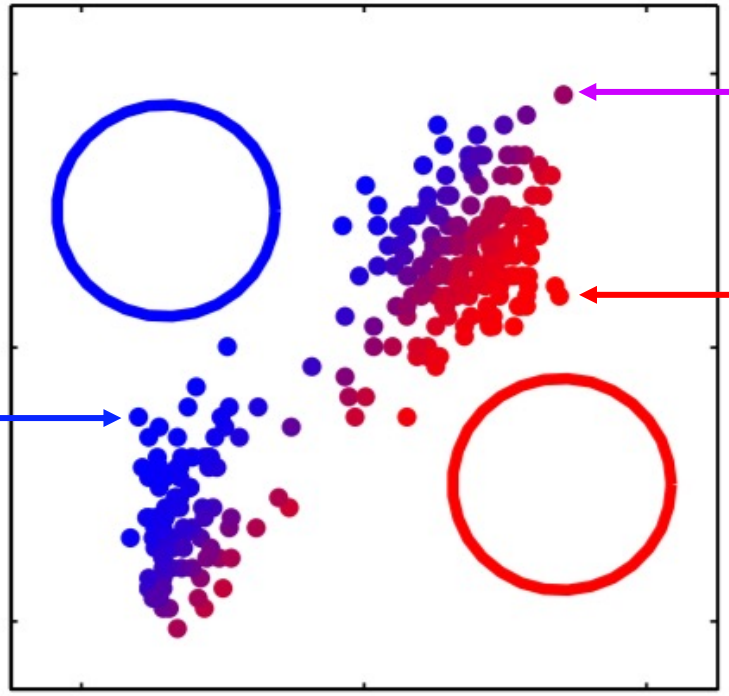
$$= \sum_{i=1}^{N} \frac{\nabla_{\boldsymbol{\mu}_k}[\ \pi_k \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\ ]}{\sum_j \pi_j \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

$$= \sum_{i=1}^{N} \frac{\pi_k \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_j \pi_j \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_k)$$

But first, what is this ratio about?

Interlude: Introduce "responsibilities"

- For a given data point $\mathbf{x}$, there may be uncertainty about which component is "responsible" for $\mathbf{x}$

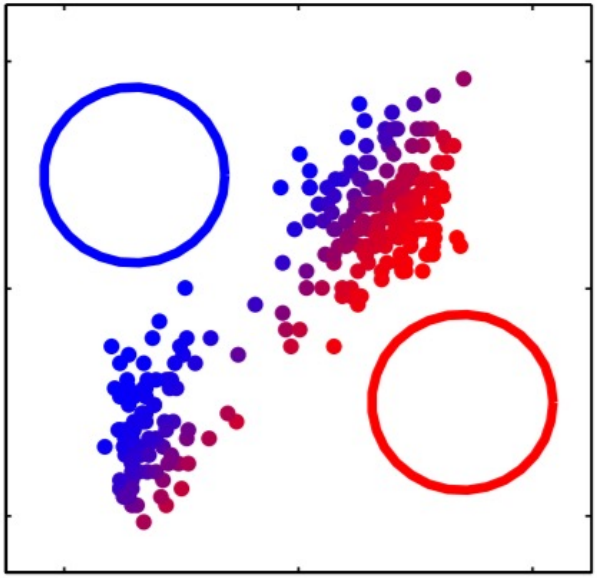


GMM is "soft assignment" whereas *K*-means is "hard assignment"

Image credit: Christopher M. Bishop

Interlude: Introduce "responsibilities"

- The "responsibility" of a component for a data point is "the probability that component *k* generated $\mathbf{x}$"
- This quantity appears many times when deriving the EM algorithm, so we introduce shorthand:



$$\gamma(z_k) \equiv p(z_k = 1 \mid \mathbf{x})$$

(by Bayes thm) $$= \frac{p(\mathbf{x} \mid z_k = 1)p(z_k = 1)}{p(\mathbf{x})}$$

$$= \frac{\pi_k \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_j \pi_j \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

Image credit: Christopher M. Bishop

Coordinate descent on $\boldsymbol{\mu}_k$

- Set gradient to zero: $0 = \sum_{i=1}^{N} \gamma(z_{ik}) \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_k)$
- Multiply by $\boldsymbol{\Sigma}_k$ and rearrange terms to get

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma(z_{ik}) \mathbf{x}_i \quad \text{where we define } N_k \equiv \sum_{i=1}^{N} \gamma(z_{ik})$$

Think of this quantity (the denominator) as "the effective number of points in component *k*"

- Given current values for $\boldsymbol{\pi}, \boldsymbol{\Sigma}$ this is the $\boldsymbol{\mu}$ that maximizes the likelihood as much as possible!
  - Deriving optimal updates for $\boldsymbol{\pi}, \boldsymbol{\Sigma}$ is similar

The EM algorithm

- E step: evaluate each responsibility $\gamma(z_{ik})$

$$\gamma_{ik} = \frac{\pi_k \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_j \pi_j \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

- M step: update model parameters for all *k*

$$\boldsymbol{\mu}_k^{\text{new}} = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} \mathbf{x}_i$$

$$\pi_k^{\text{new}} = \frac{N_k}{N}$$

K-means only modeled the means!
Covariance and mixing weights were implicitly *constant*!

$$\boldsymbol{\Sigma}_k^{\text{new}} = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} (\mathbf{x}_i - \boldsymbol{\mu}_k^{\text{new}})(\mathbf{x}_i - \boldsymbol{\mu}_k^{\text{new}})^T$$

“Expectation” step (E-step)

γ_{ik}^{old} (not yet defined)

γ_{ik}^{new} (after 1st E-step)

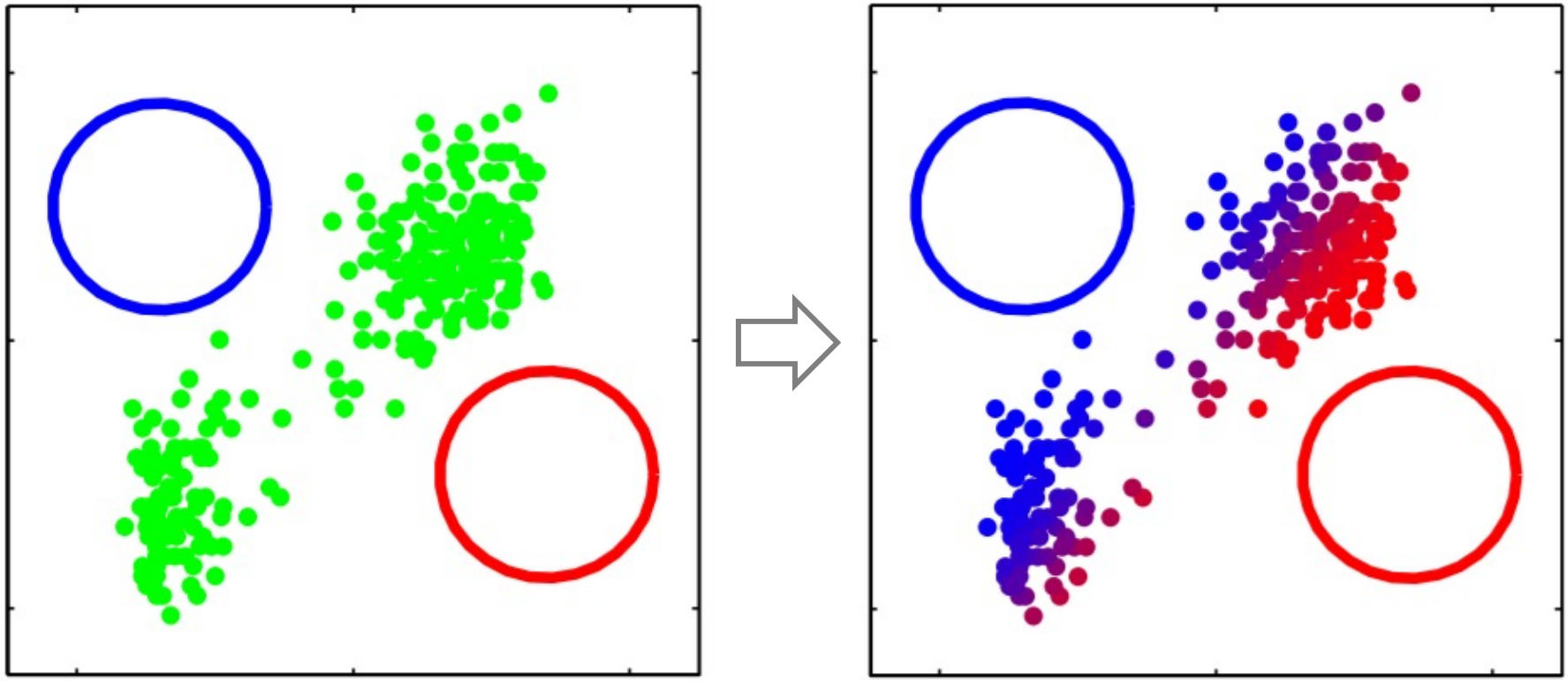
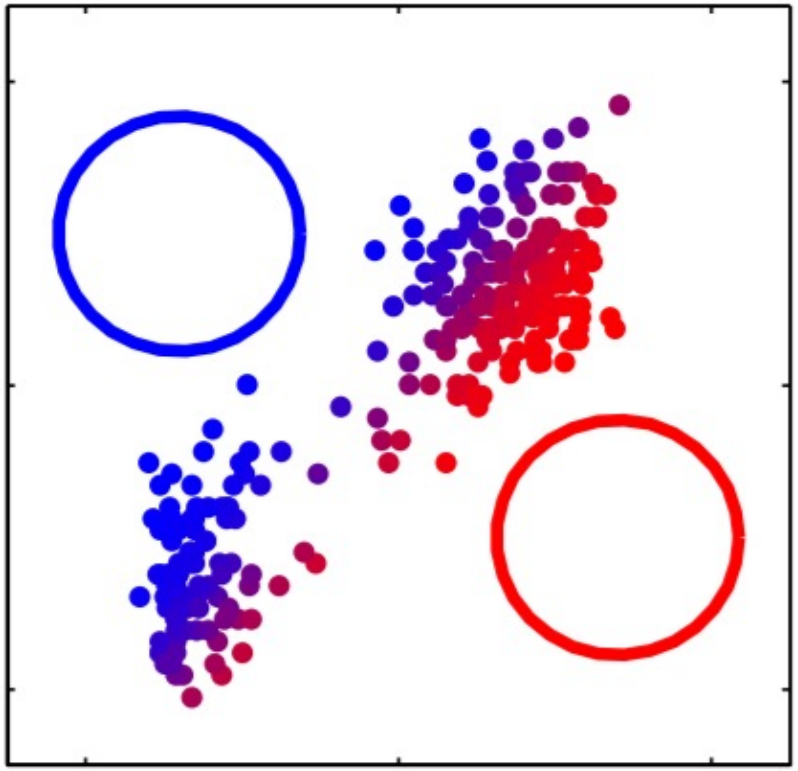


Image credit: Christopher M. Bishop

"Maximization" step (M-step)

$\pi^{\text{old}}, \mu^{\text{old}}, \Sigma^{\text{old}}$



$\pi^{\text{new}}, \mu^{\text{new}}, \Sigma^{\text{new}}$

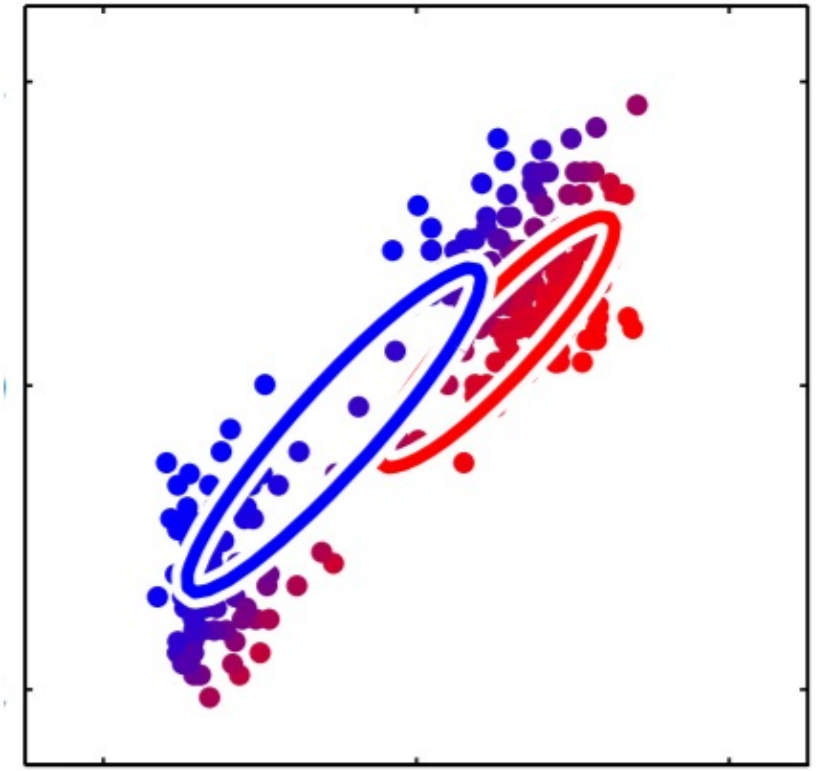


Image credit: Christopher M. Bishop

EM algorithm versus *K*-means

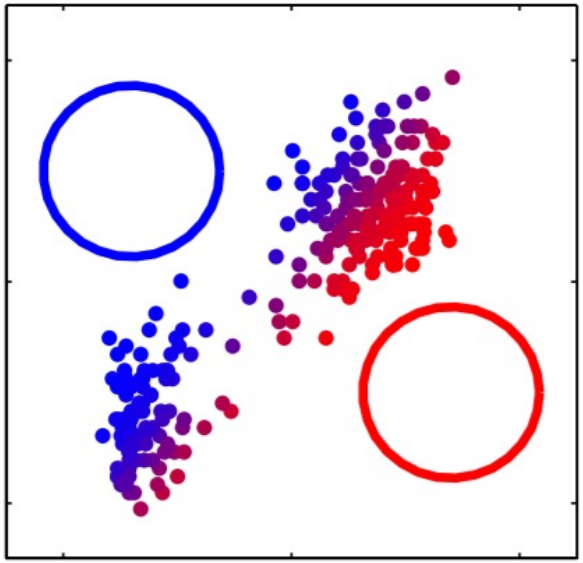
- E step:

$$\gamma_{ik} = \frac{\pi_k \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_j \pi_j \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

"$\mathbf{x}_i$ well-explained by $\mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$?"

$$r_{ik} = \begin{cases} 1 & \text{if } k = \arg\min_j \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2 \\ 0 & \text{otherwise.} \end{cases}$$

"$\mathbf{x}_i$ closest to $\boldsymbol{\mu}_k$? (yes/no)"



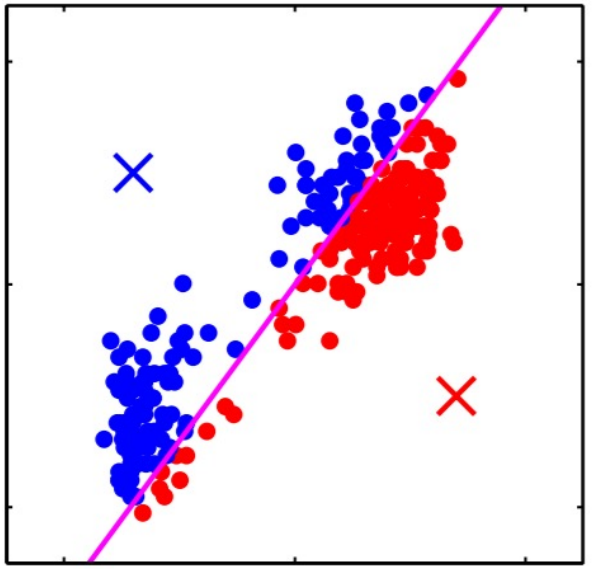


Image credit: Christopher M. Bishop

EM algorithm versus *K*-means

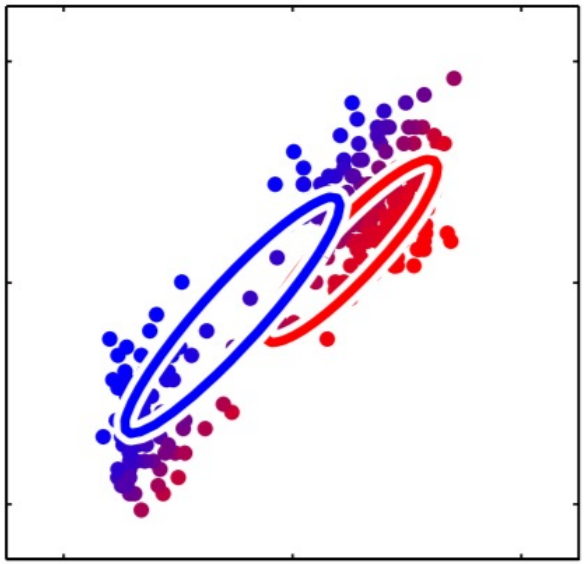
- M step:

$$\boldsymbol{\mu}_k = \frac{\sum_{i=1}^{N} \gamma_{ik} \mathbf{x}_i}{\sum_{i=1}^{N} \gamma_{ik}}$$

"weighted mean of all $\mathbf{x}_i$"

$$\boldsymbol{\mu}_k = \frac{\sum_{i=1}^{N} r_{ik} \mathbf{x}_i}{\sum_{i=1}^{N} r_{ik}}$$

"mean of $\mathbf{x}_i$ assigned to k"



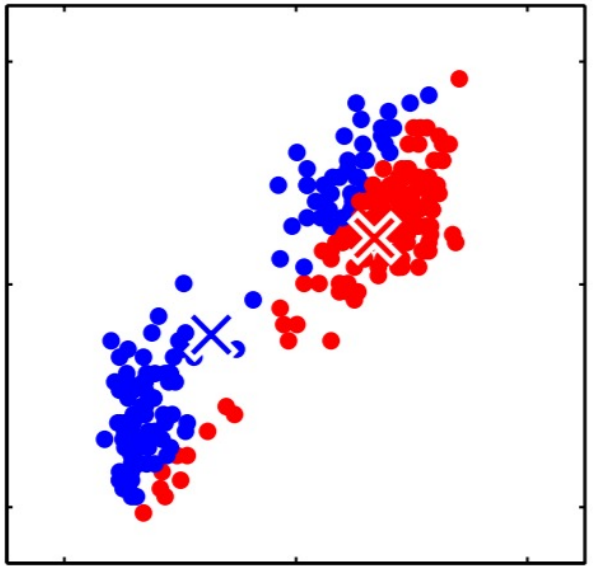
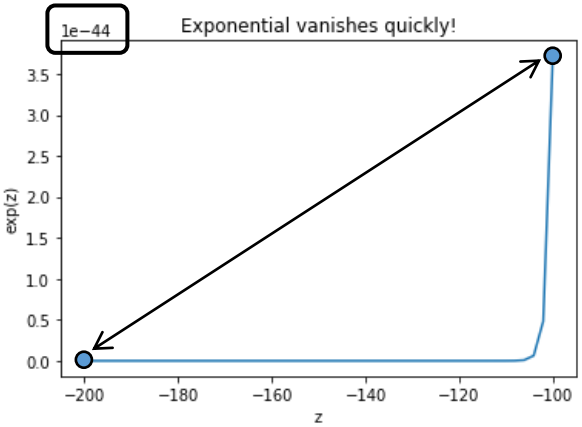


Image credit: Christopher M. Bishop

K-means as special case of EM

- Can derive *K*-means from EM algorithm by...
  - Assuming all $\pi_k = \frac{1}{K}$ (fixed equal mixture weights)
  - Assuming all $\boldsymbol{\Sigma}_k = \sigma^2 \mathbf{I}$ (fixed isotropic variance)
  - Letting $\sigma \to 0$ (standard deviation approaches zero)
  - Then $\lim_{\sigma \to 0} \gamma_{ik} = r_{ik}$ because $\gamma_{ik} = \frac{\exp\{-\frac{1}{2\sigma^2} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2\}}{\sum_j \exp\{-\frac{1}{2\sigma^2} \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2\}}$



As $\sigma \to 0$ the term *j* for which $\|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2$ is smallest (argmin) will dominate (will be much larger than all the other terms!)

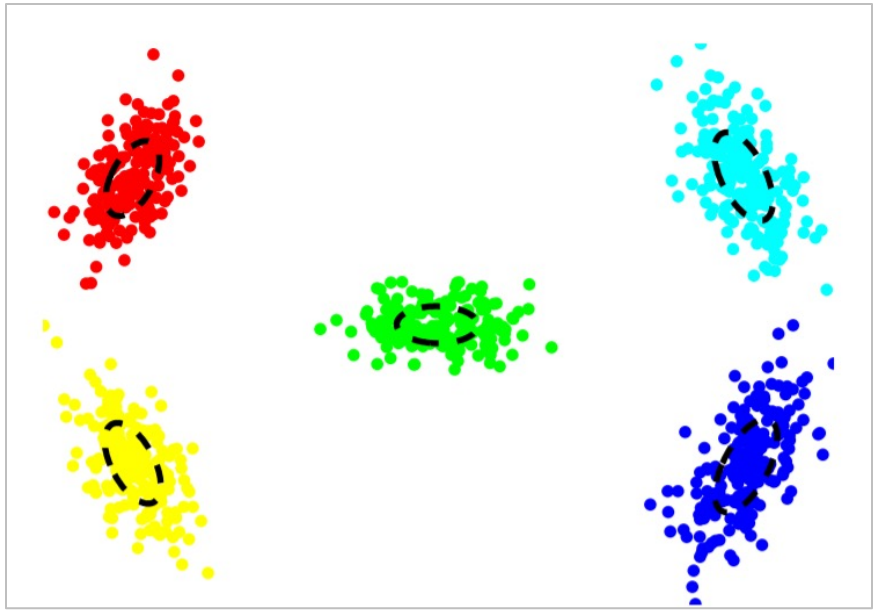
$$\frac{\exp(-100)}{\exp(-100) + \exp(-200)} \approx 1.0 \qquad \frac{\exp(-200)}{\exp(-100) + \exp(-200)} \approx 0.0$$

This is a **softmax** function! Important in multi-class classification

GMM versus *K*-means

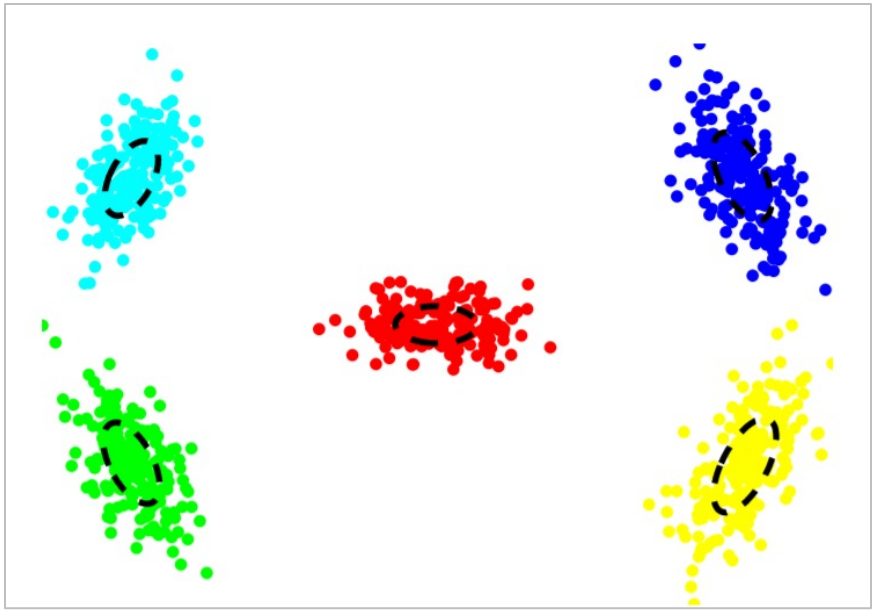
- Both work well for non-overlapping ball-like clusters

EM algorithm for GMMs



K=5

weighted elliptical *K*-means algorithm

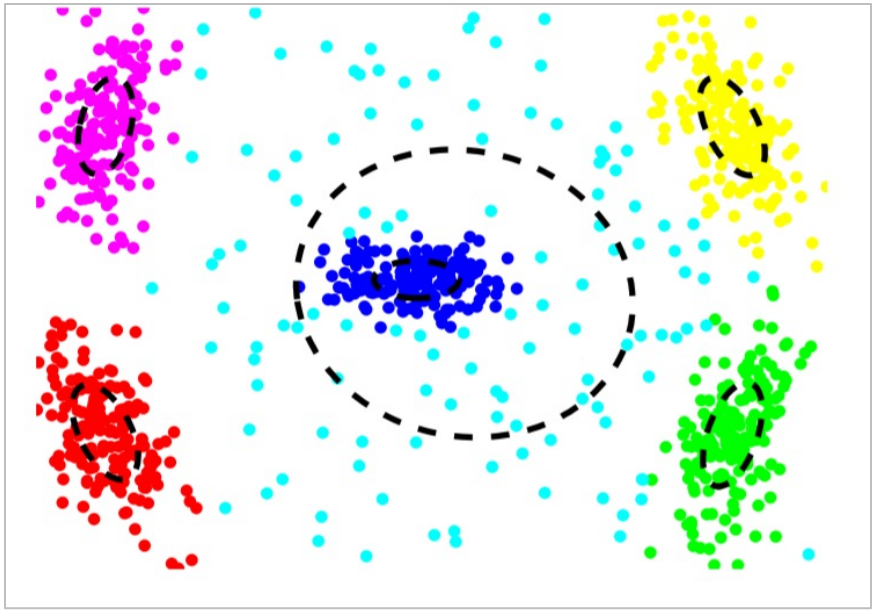


K=5

GMM versus *K*-means

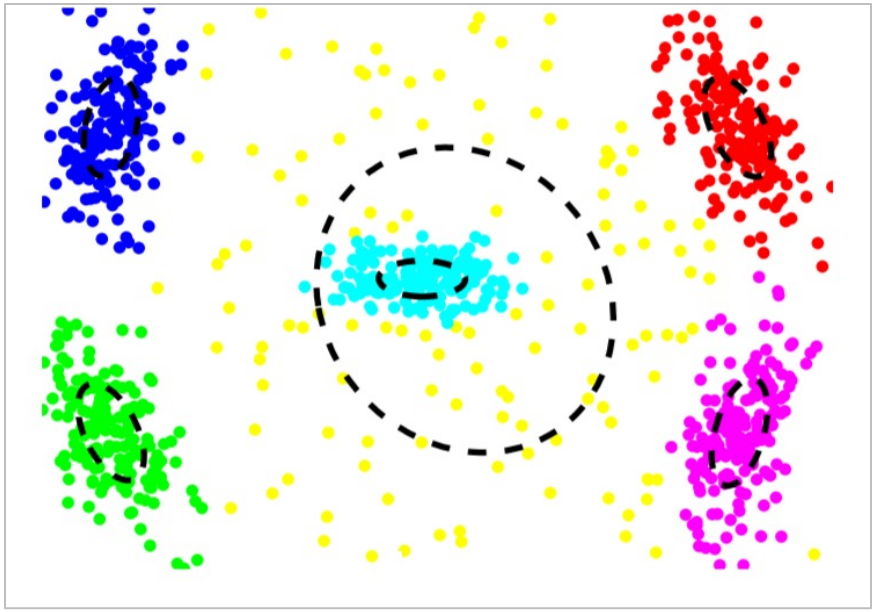
- Both can handle outliers to some degree

EM algorithm for GMMs



K=6

weighted elliptical *K*-means algorithm

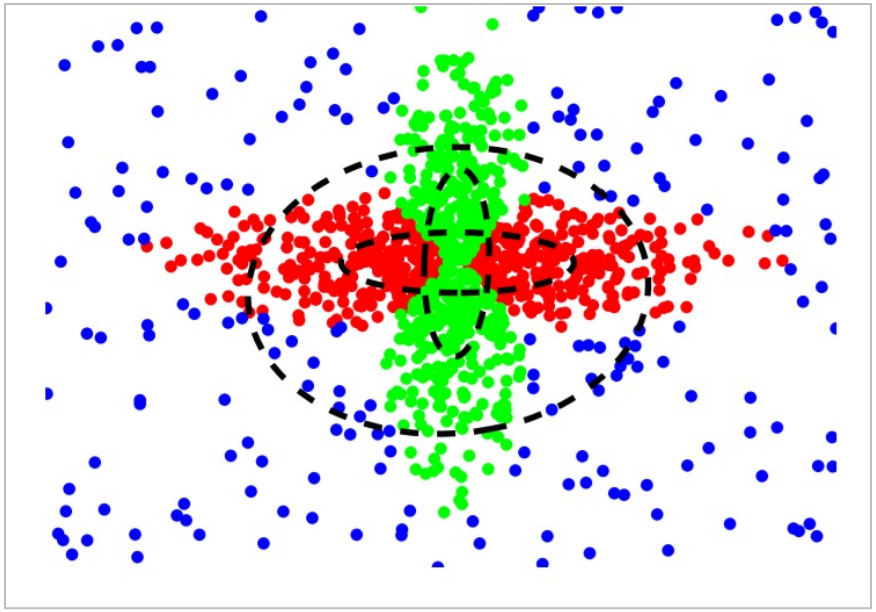


K=6

GMM versus *K*-means

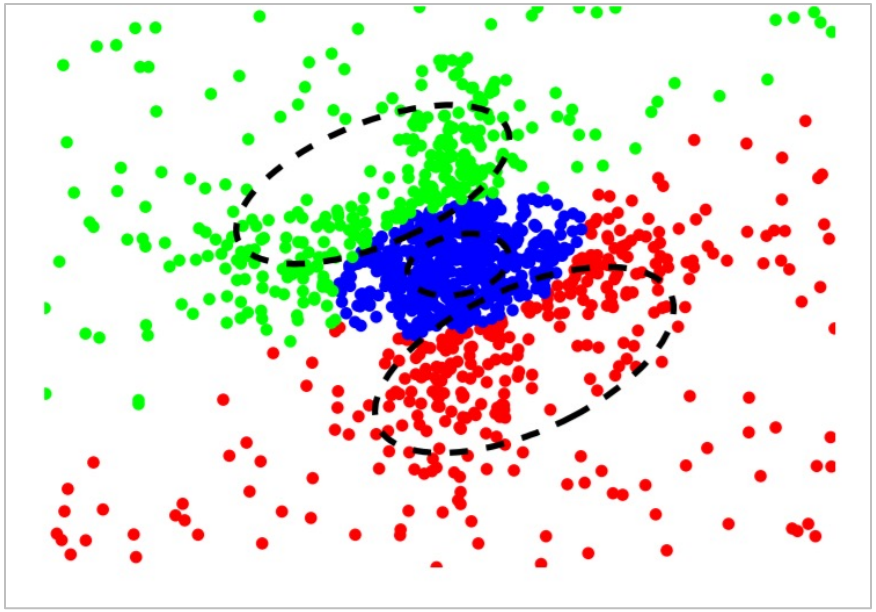
- EM works better for overlapping structures

EM algorithm for GMMs



K=3

weighted elliptical *K*-means algorithm

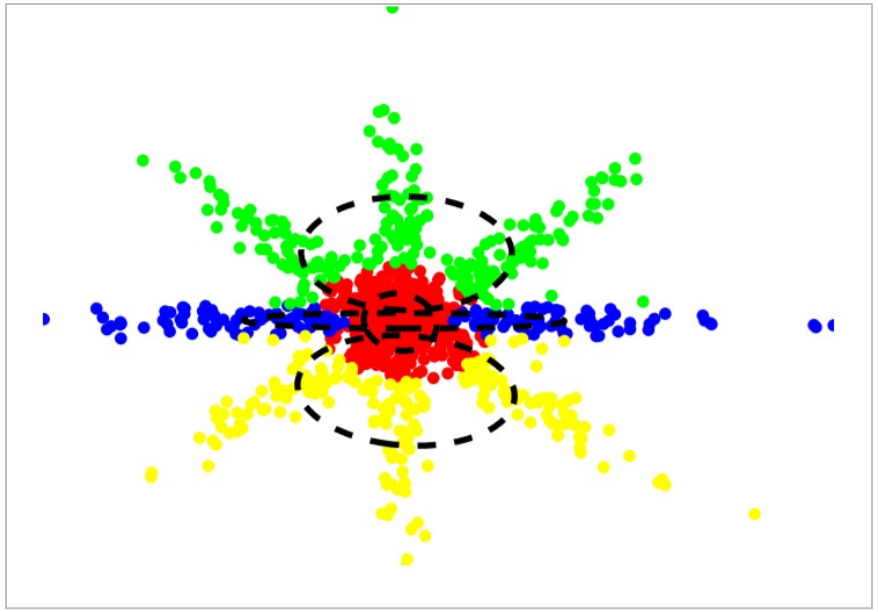


K=3

GMM versus *K*-means

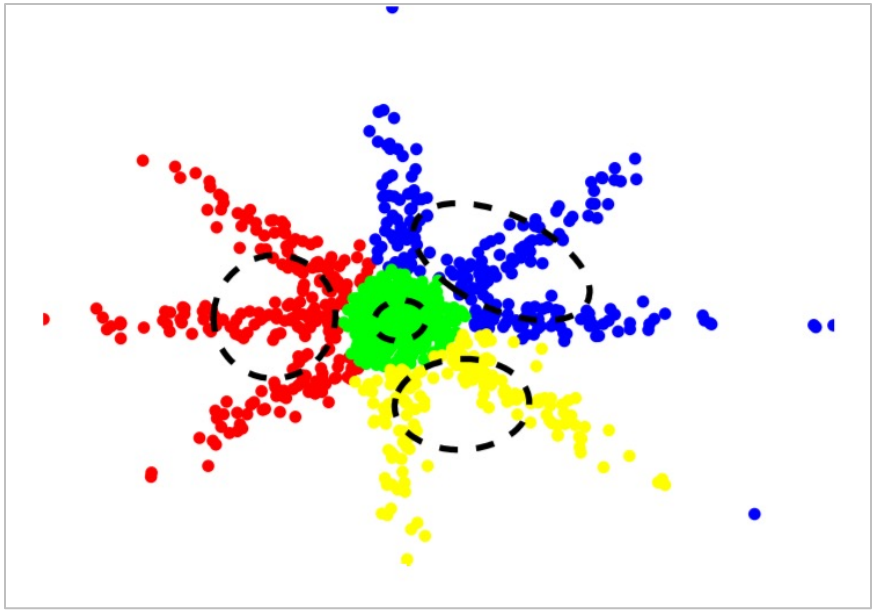
- But both can get stuck in local minima

EM algorithm for GMMs



K=4

weighted elliptical *K*-means algorithm

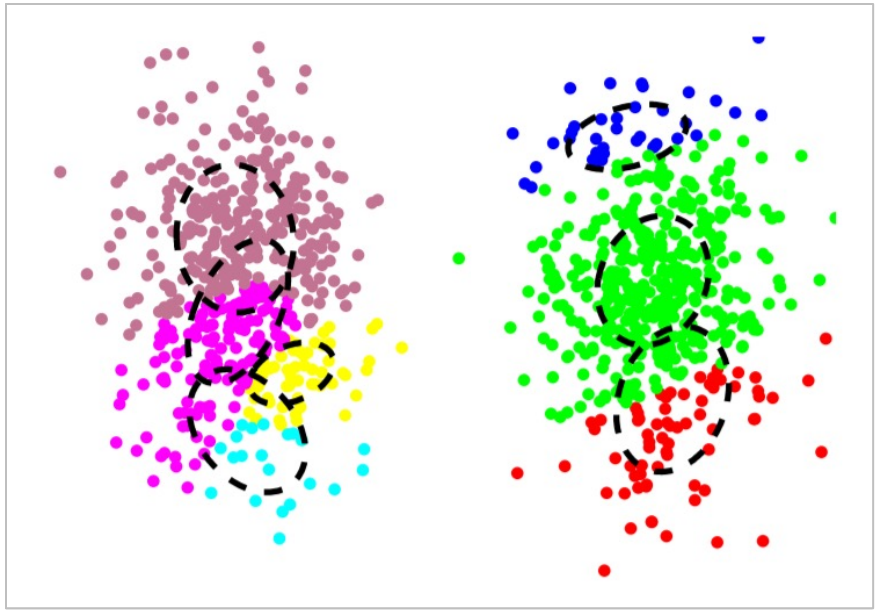


K=4

GMM versus *K*-means

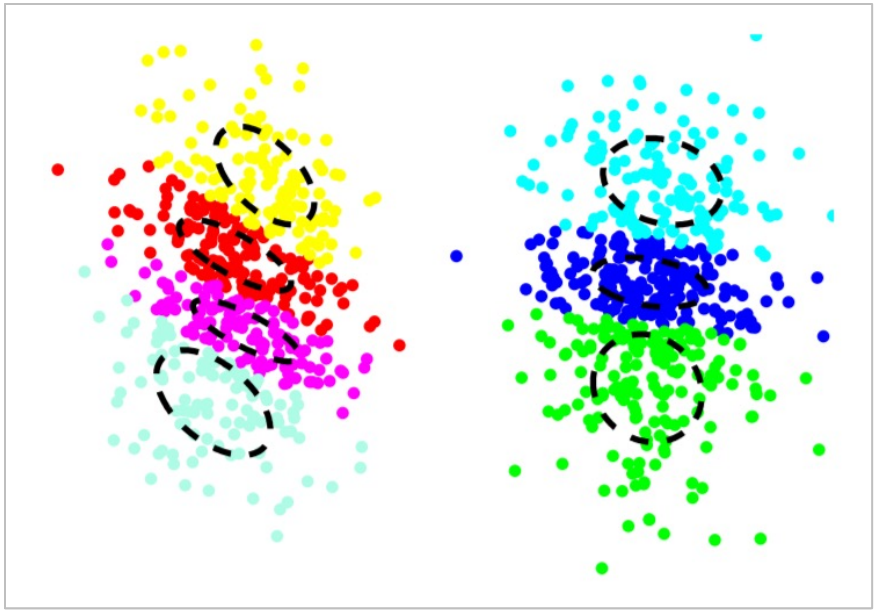
- And both sensitive to guessing number of components

EM algorithm for GMMs



K=7

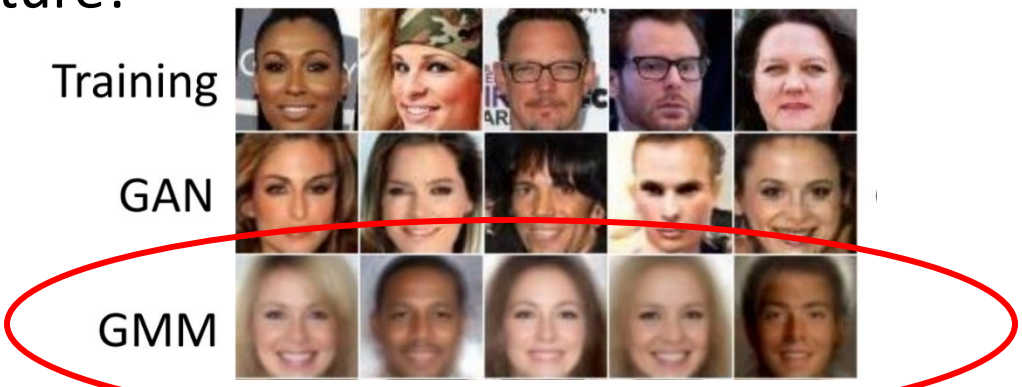
weighted elliptical *K*-means algorithm



K=7

GMM versus *K*-means

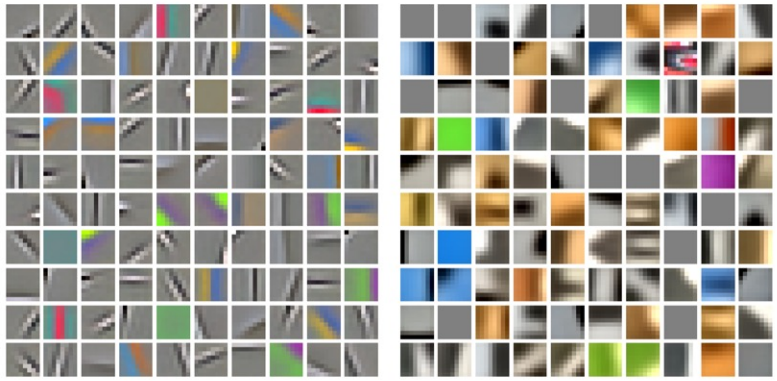
- K-means tends to converge much faster
- Common practice:
  1. Run K-means to provide "smart" initialization to GMM
  2. Run EM algorithm to refine GMM parameters
- Gaussian mixtures are *generative* models
  - After fitting GMM to training data, can sample more data with similar structure!



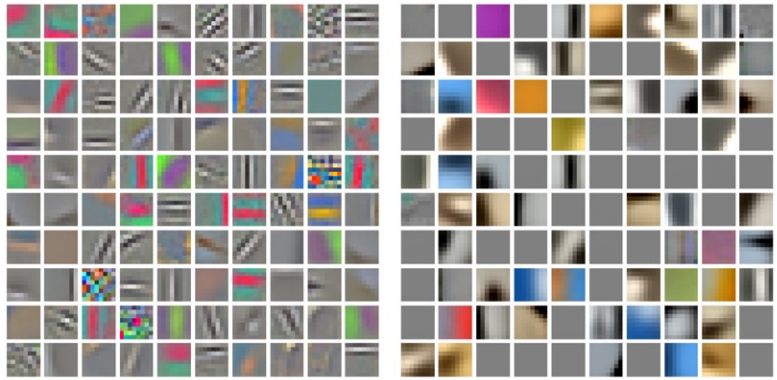
Richardson & Weiss 2018 https://arxiv.org/abs/1805.12462

Clustering as feature learning in ML

Idea: unsupervised "feature learning" by clustering patches from many unlabeled images



(a) K-means (with and without whitening)



(b) GMM (with and without whitening)

Figure 2: Randomly selected bases (or centroids) trained on CIFAR-10 images using different learning algorithms.

Then, for supervised classification task, convert pixels into "patch relevance" scores and classify based on *that* instead!

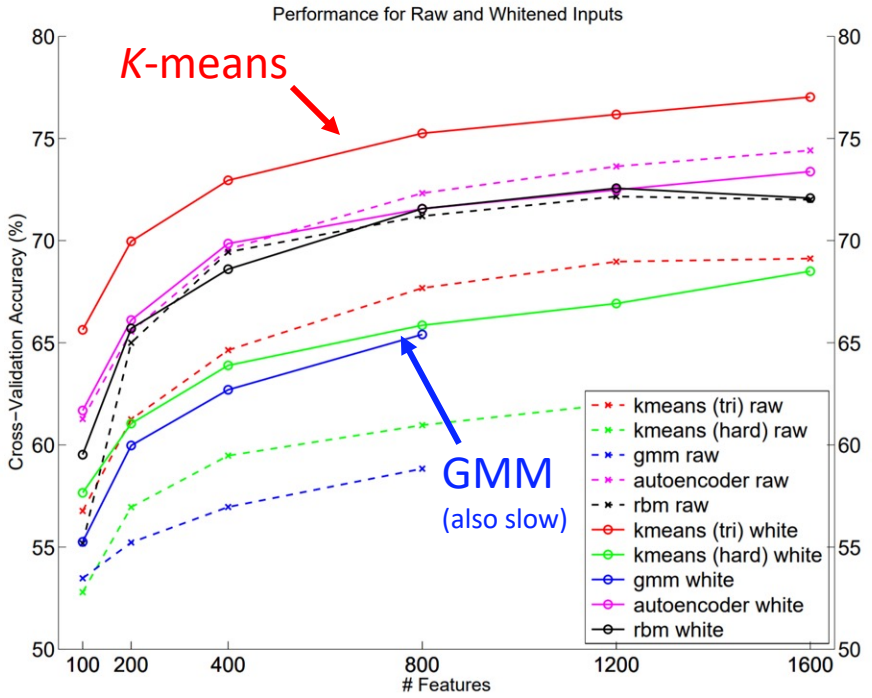


Figure 3: Effect of whitening and number of bases (or centroids).

>2700 citations 🤯

Adam Coates, Honglak Lee, Andrew Y. Ng

http://proceedings.mlr.press/v15/coates11a/coates11a.pdf

Clustering as feature learning in ML

Idea: unsupervised "feature learning" by clustering patches from many unlabeled images

"**K-means clustering:** We apply K-means clustering to learn K centroids $c^{(k)}$ from the input data. Given the learned centroids $c^{(k)}$, we consider two choices for the feature mapping f. The first is the standard 1-of-K, hard-assignment coding scheme:

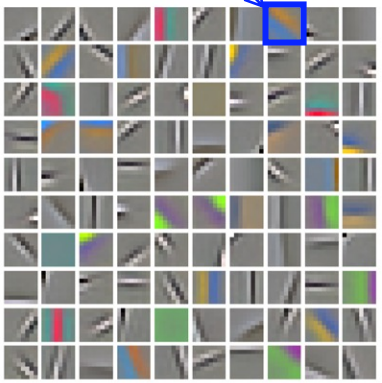
$$f_k(x) = \begin{cases} 1 & \text{if } k = \arg\min_j ||c^{(j)} - x||_2^2 \\ 0 & \text{otherwise.} \end{cases}$$

This is a (maximally) sparse representation that has been used frequently in computer vision [5]."

sparse 1-of-100 feature vector

$$f(\mathbf{x}) = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \end{bmatrix}^T$$

32x32 patch
= 1024 raw pixels



"**Gaussian mixtures:** Gaussian mixture models (GMMs) represent the density of input data as a mixture of K Gaussian distributions and is widely used for clustering. GMMs can be trained using the Expectation-Maximization (EM) algorithm as in [1]. We run a single iteration of K-means to initialize the mixture model.[4] The feature mapping f maps each input to the posterior membership probabilities:

$$f_k(x) = \frac{1}{(2\pi)^{d/2}|\Sigma_k|^{1/2}} \cdot \exp\left(-\frac{1}{2}(x - c^{(k)})^\top \Sigma_k^{-1}(x - c^{(k)})\right)$$

where Σ_k is a diagonal covariance and ϕ_k are the cluster prior probabilities learned by the EM algorithm."

soft and mostly-sparse 100-dim feature vector

$$f(\mathbf{x}) = \begin{bmatrix} 0.1 & 0.6 & 0.3 & \dots & 0 \end{bmatrix}^T$$

In short: Patch contents get summarized by associating to 'meaningful' centroid(s) that were learned ahead of time!

Adam Coates, Honglak Lee, Andrew Y. Ng
http://proceedings.mlr.press/v15/coates11a/coates11a.pdf

Impact of K-Means and EM

[PDF] Least squares quantization in PCM

S Lloyd - IEEE transactions on information theory, 1982 - sites.cs.ucsb.edu

It has long been realized tha
signals to handle, the quantu
where the signal amplitude is
in the limit as the number of
quanta per unit voltage shou
voltage of signal amplitudes.

☆ 99 Cited by 10640 F

Some methods for classification and analysis of multivariate observations

J MacQueen - Proceedings of the fifth Berkeley symposium on …, 1967 - books.google.com

The main purpose of this paper is to describe a process for partitioning an A^-dimensional population into k sets on the basis of a sample. The process, which is called'fc-means,'appears to give partitions which are reasonably efficient in the sense of within-class variance. That is, if p is the probability mass function for the population, S={Si, S2,••■, Sk} is a partition of En, and Ui, i—1, 2,•••, k, is the conditional mean of p over the set Si, then w2 (S)= H*-i jsi\z—U;\2dp (z) tends to be low for the partitions S generated by the method. We …

☆ 99 Cited by 25235 Related articles All 20 versions

Maximum Likelihood from Incomplete Data Via the ***EM*** **Algorithm**

AP **Dempster**, NM Laird… - Journal of the Royal …, 1977 - Wiley Online Library

A broadly applicable algorithm for computing maximum likelihood estimates from incomplete data is presented at various levels of generality. Theory showing the monotone behaviour of the likelihood and convergence of the algorithm is derived. Many examples are sketched …

★ 99 Cited by 55860 Related articles All 59 versions

PRML Readings

§9.0.0 Mixture Models and EM

§9.1.0 *K*-means Clustering

§9.1.1 Image segmentation and compression

§9.2.0 Mixtures of Gaussians

§9.2.1 Maximum likelihood (of Mixtures of Gaussians)

§9.2.2 EM for Gaussian mixtures

§9.3.2 Relation to *K*-means

PRML Errata (error in GMM gradient!)

Page 434 Equation (9.15): σ_j should be σ_j^D in the denominator on the r.h.s.

Page 435 Third paragraph, line 3: "will play" and "discuss" should be "played" and "discussed', respectively.'

Page 435 Equation (9.16): There are a matrix inverse missing and an extra minus ('$-$') sign on the r.h.s.; the correct form is

$$0 = \sum_{n=1}^{N} \underbrace{\frac{\pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_j \pi_j \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}}_{\gamma(z_{nk})} \boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_k).$$

Page 435 Line 3 after Equation (9.16): $\boldsymbol{\Sigma}_k^{-1}$ should be $\boldsymbol{\Sigma}_k$, i.e., no inverse.