



ARISTOTLE UNIVERSITY OF THESSALONIKI



FACULTY OF ENGINEERING

Pattern Recognition & Machine Learning

Unsupervised Learning

Panagiotis C. Petrantonakis

Assistant Professor

Dept. of Electrical and Computer Engineering

ppetrant@ece.auth.gr

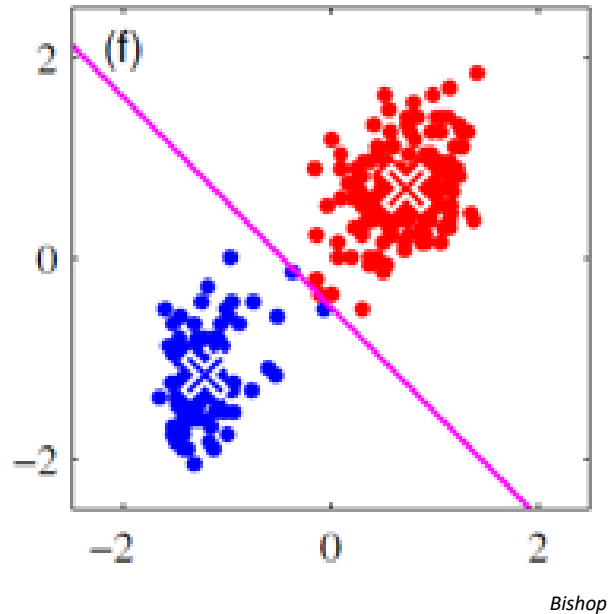
Fall Semester

Supervised vs. Unsupervised learning

- So far, we have seen supervised learning approaches:
 - Data: $D = \{\mathbf{X}, \mathbf{T}\}$
 - Goal: $f \approx t, p(\omega_i | \mathbf{x})$
 - Classification (discrete outcome) or regression (continuous outcome)
- Unsupervised learning:
 - Data: $D = \{\mathbf{X}\}$
 - Goal: $p(\mathbf{z} | \mathbf{x})$
 - clustering (discrete), dimensionality reduction (continuous)
- \mathbf{z} latent variable: infer structure of the data depending on latent variables.
- Examples:
 - Real estate example: different properties of houses may lead to assumption of different locations
 - Dimensionality reduction: how reducing dimension can also lead to the same inference

Examples

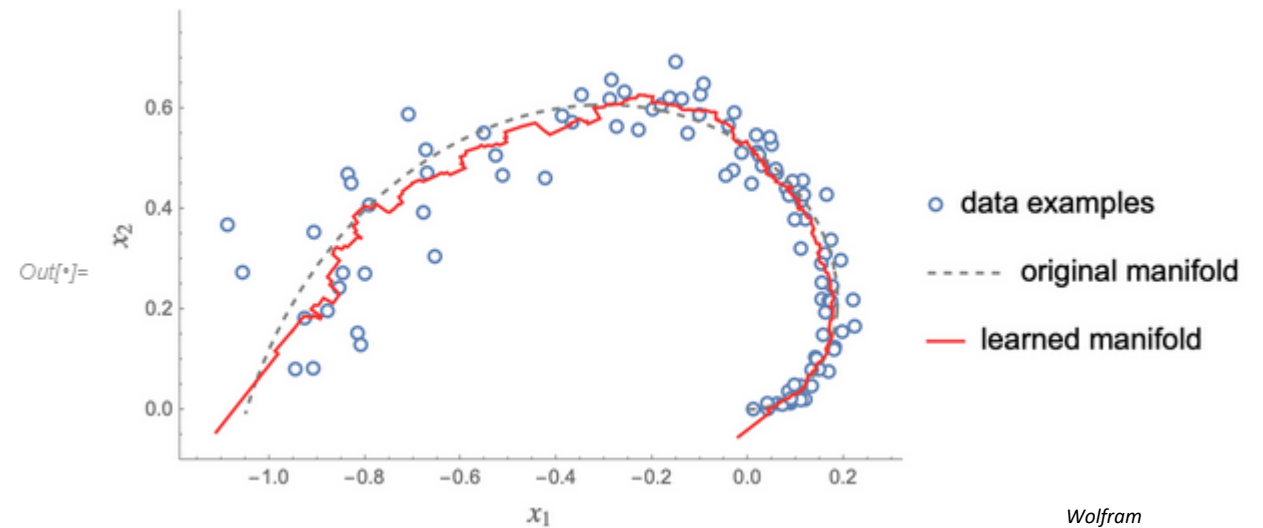
Clustering



discrete latent variables, i.e., dog and cat

Dimensionality Reduction

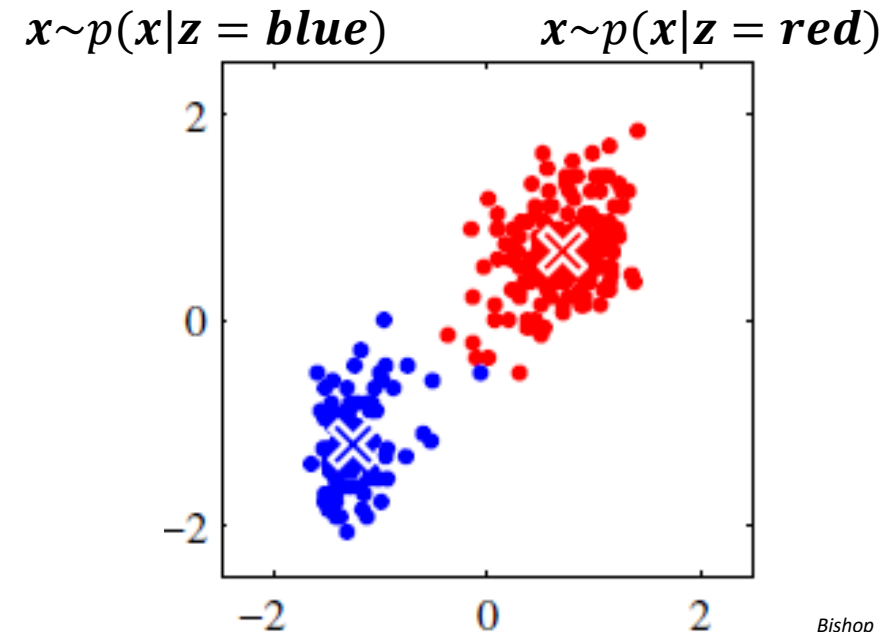
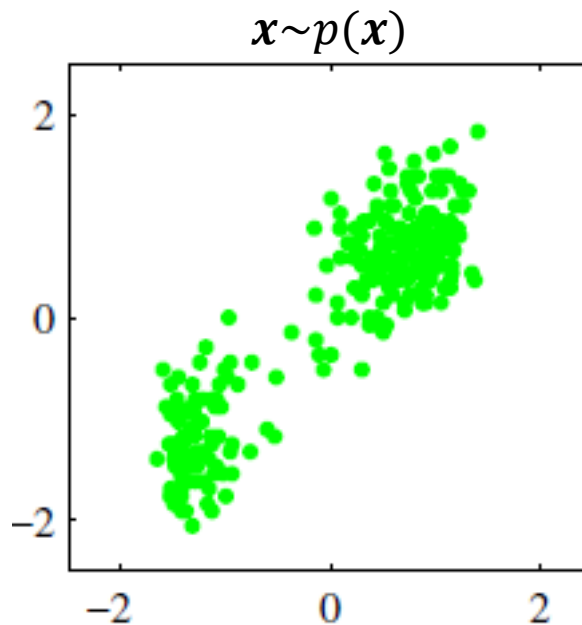
continuous latent variables, i.e., 2D data represented by 1D line.



Clustering

Clustering using k-Means

- Data are comprised of points x without any target t
- Our goal is to assign every single data point to a cluster
 - cluster: the discrete latent variable $z, z = \{cluster\ 1, cluster\ 2, \dots\}$
 - $p(x|cluster\ 1), p(x|cluster\ 2) \dots$
 - k-means does not take into consideration the probabilistic interpretation of the problem.



Error function minimization for k-Means

- Dataset: $X = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}, \mathbf{x}_i \in \mathbb{R}^D$
- We will assign each sample in one cluster (out of K) considering the minimization of:

$$J = \sum_{n=1}^N \sum_{k=1}^K z_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$$

where $z_{nk} \in \{0,1\}$ (one-hot encoding), e.g., $\mathbf{z}_n = [0 \ 0 \ 0 \ \dots \ 1 \ \dots \ 0 \ 0]$ and $\boldsymbol{\mu}_k \in \mathbb{R}^D$ are the cluster means

- K is fixed a priori!

k-Means Algorithm

- Initialize: $\boldsymbol{\mu}_k \in \mathbb{R}^D$

- **Do** until convergence:

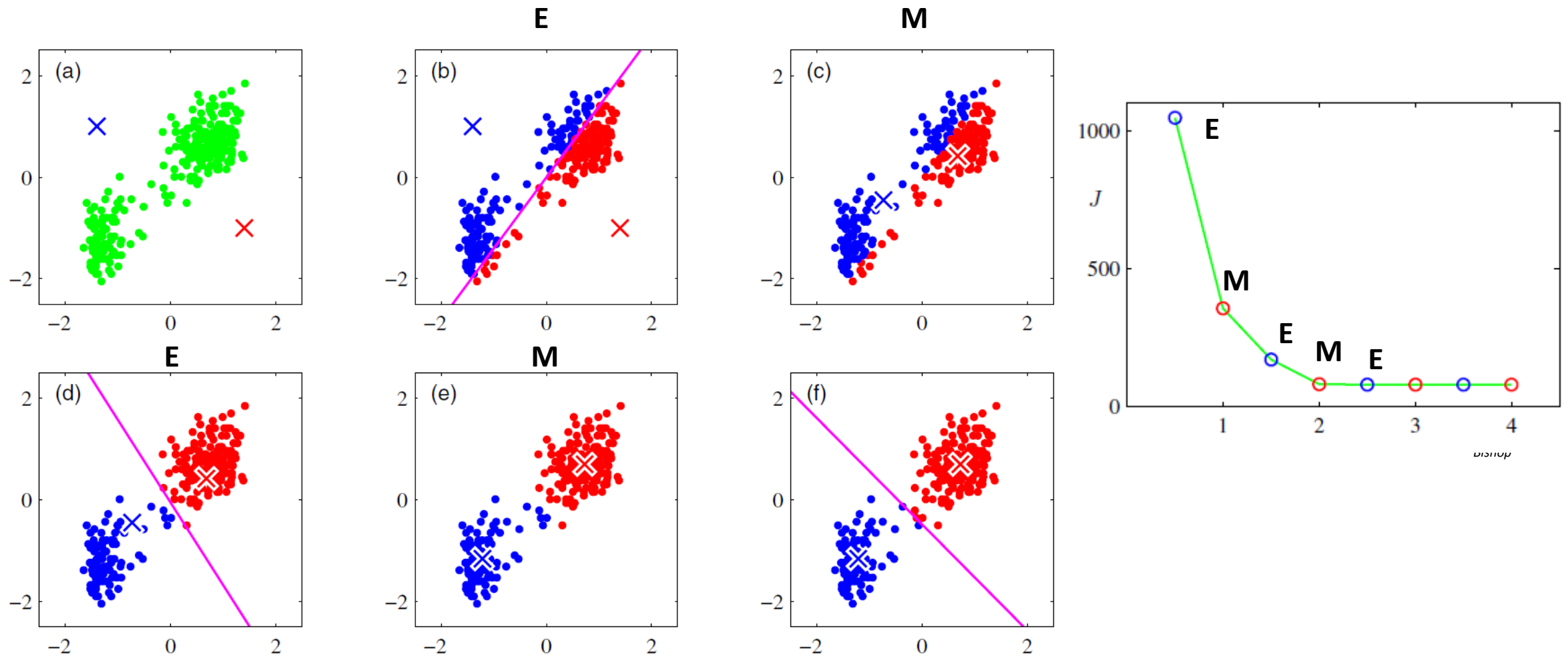
- Assign (E-step):

$$z_{nk} = \begin{cases} 1 & \text{if } k = \operatorname{argmin}_j \|\mathbf{x}_n - \boldsymbol{\mu}_j\|^2 \\ 0 & \text{otherwise} \end{cases}$$

- New means (M-step):

$$\boldsymbol{\mu}_k = \frac{\sum_n z_{nk} \mathbf{x}_n}{\sum_n z_{nk}}$$

Example



M-step derivation

$$J = \sum_{n=1}^N \sum_{k=1}^K z_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$$

- J is a convex function in $\boldsymbol{\mu}_k$.
- Minimum w.r.t. $\boldsymbol{\mu}_k$:

$$\frac{\partial}{\partial \boldsymbol{\mu}_k} \sum_{n=1}^N \sum_{j=1}^K z_{nj} \|\mathbf{x}_n - \boldsymbol{\mu}_j\|^2 = \sum_{n=1}^N z_{nk} \frac{d}{d \boldsymbol{\mu}_k} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2 = -2 \sum_{n=1}^N z_{nk} (\mathbf{x}_n - \boldsymbol{\mu}_k)^T$$

- Άρα $\sum_{n=1}^N z_{nk} \mathbf{x}_n - \boldsymbol{\mu}_k \sum_{n=1}^N z_{nk} = 0 \Rightarrow \boldsymbol{\mu}_k = \frac{\sum_{n=1}^N z_{nk} \mathbf{x}_n}{\sum_{n=1}^N z_{nk}}$

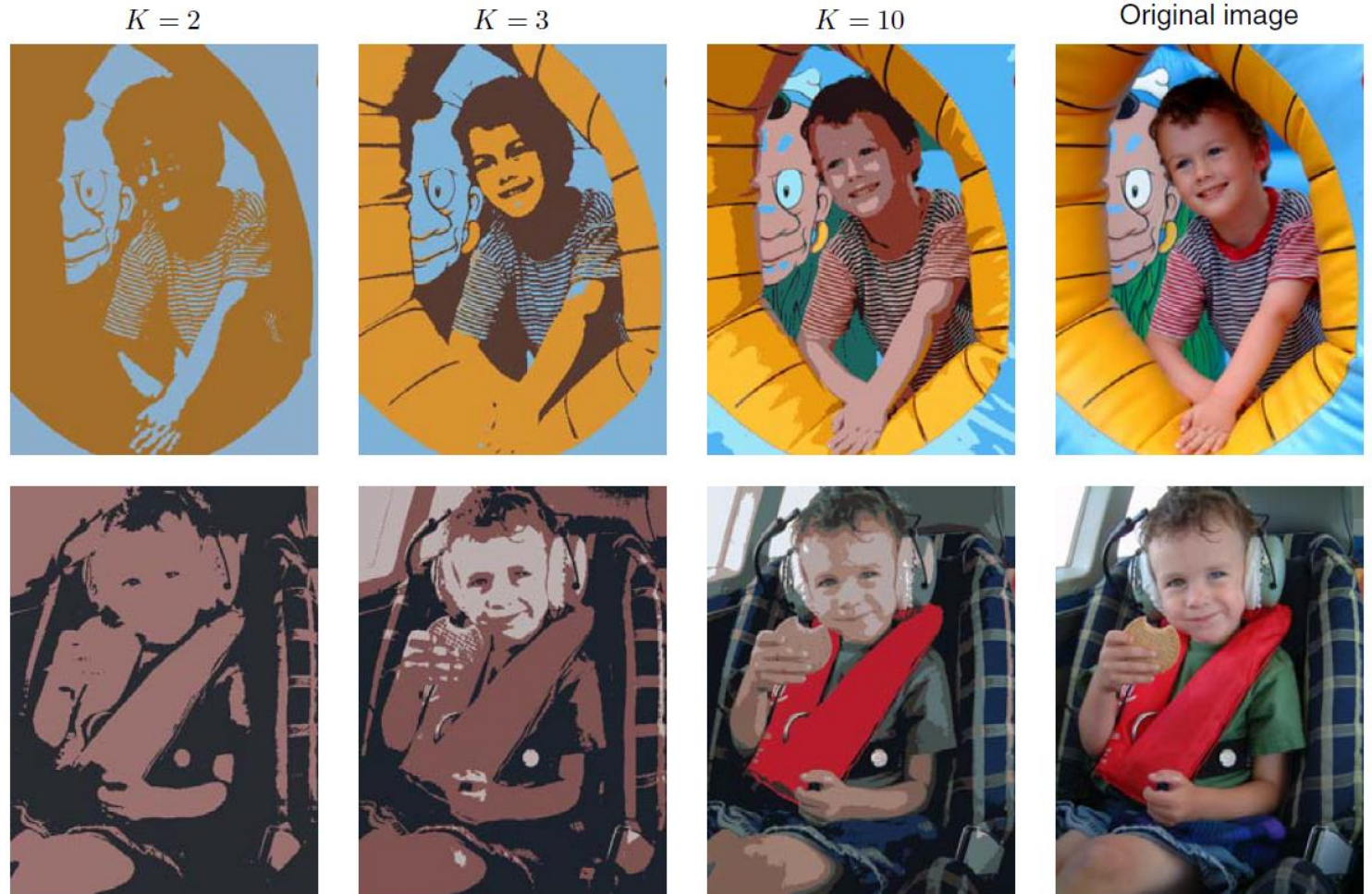
Convergence of k-Means

- Unfortunately, $J = \sum_{n=1}^N \sum_{k=1}^K z_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$ is non convex for z_{nk} and $\boldsymbol{\mu}_k$ together.
- It is probable that k-Means converges to local minimum.
- **Solution:**
 - Random restarts with different initializations of $\boldsymbol{\mu}_k$
 - Finally select realization with minimal J

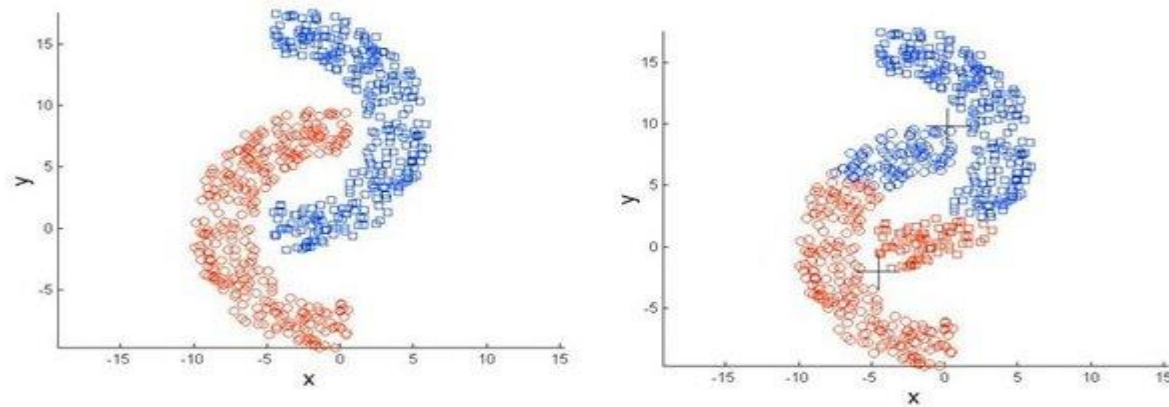
Application

- **Compression:** Instead of storing all RGB values in the range $[0, 255]$ for each pixel we just store the cluster number each pixel belongs, K = number of colors.
- **Segmentation:** Pixels that are of similar color they tend to be in the same segment (though it does not take any spatial relations into account)

Data point: x_n is one pixel $[R, G, B]$



Where k-Means fails



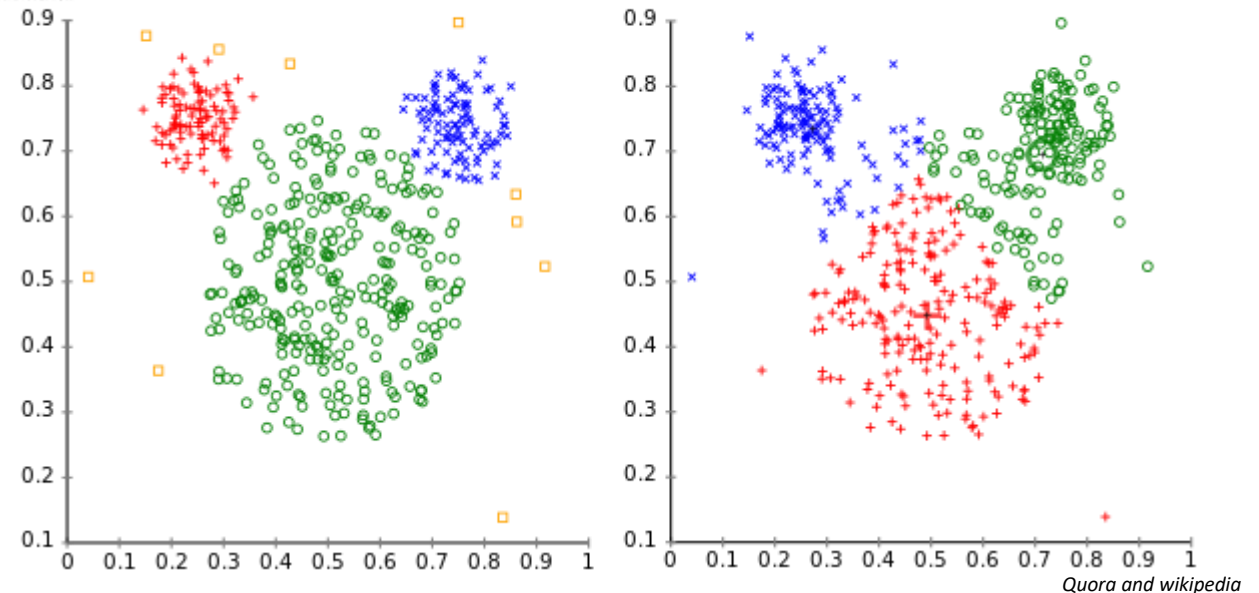
Original Points

K-means (2 Clusters)

Only generates spherical clusters

Original Data

k-Means Clustering



There is no way to define cluster size

How can we improve k-Means?

$$J = \sum_{n=1}^N \sum_{k=1}^K z_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$$

- What do we usually use if we have an error function that depends on single sample errors:

How can we improve k-Means?

$$J = \sum_{n=1}^N \sum_{k=1}^K z_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$$

- What do we usually use if we have an error function that depends on single sample errors:
- SGD for big data implementations:

$$\boldsymbol{\mu}_k = \boldsymbol{\mu}_k - \eta \frac{\partial}{\partial \boldsymbol{\mu}_k} J = \boldsymbol{\mu}_k + 2\eta(\mathbf{x}_n - \boldsymbol{\mu}_k)$$

How can we improve k-Means?

- We can use other (dis) similarity measures rather than the Euclidean distance:

$$J = \sum_{n=1}^N \sum_{k=1}^K z_{nk} \mathcal{L}(\mathbf{x}_n, \boldsymbol{\mu}_k)$$

- $\mathcal{L}(\mathbf{x}_n, \boldsymbol{\mu}_k)$ can be any other distance measure. This will alleviate the problem of Euclidean distance with the outliers.
- We can also use other ways to define the center of the cluster, e.g., actual sample instead of the mean which may be or may not be a sample of the dataset.
- The above family of algorithms are called k-Medoids.

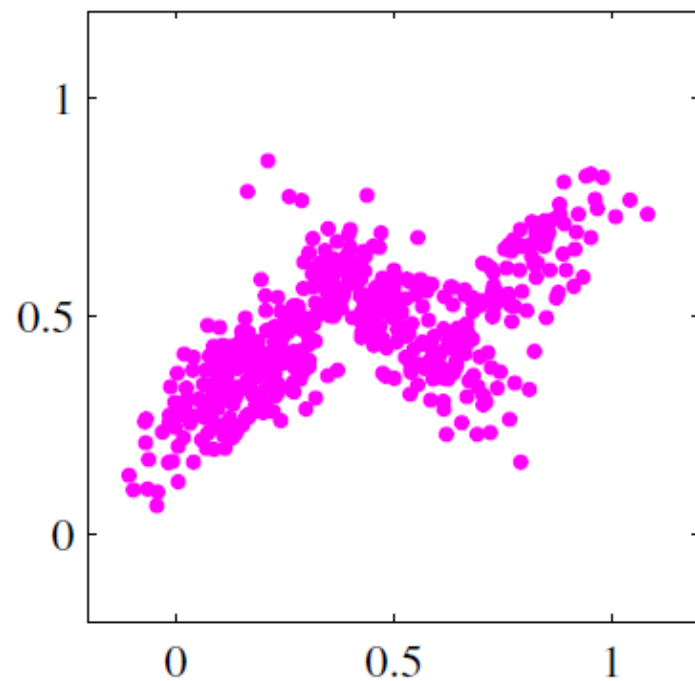
Pros and Cons

- Pros:
 - simple
 - Fast
- Cons:
 - Local minima (it may converge to one of them)
 - Only “spherical” clusters
 - Sensitive to the features scale
 - Number of clusters has to be selected in advance
 - Cluster assignments are “hard”, i.e., they do not take into account a **probabilistic perspective**
- Next topic: Gaussian Mixture Models
 - “soft” assignment
 - probabilistic interpretation

Gaussian Mixture Model (GMM)

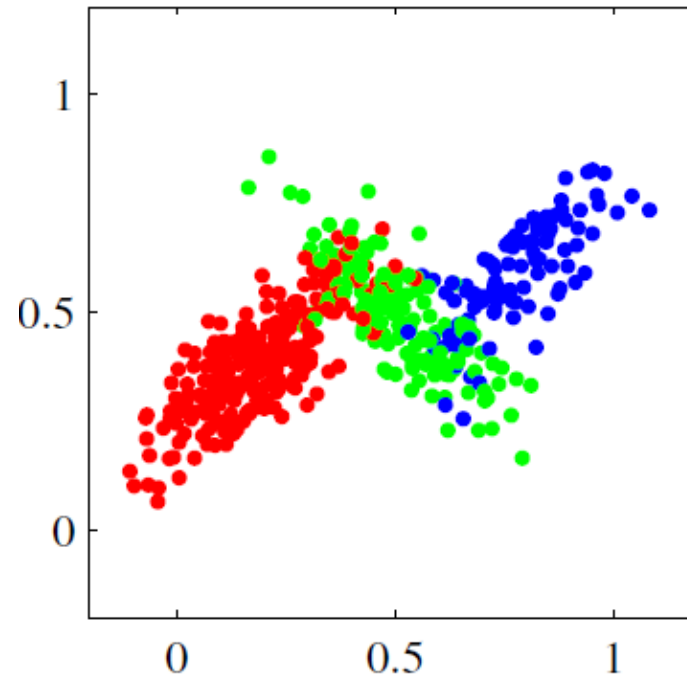
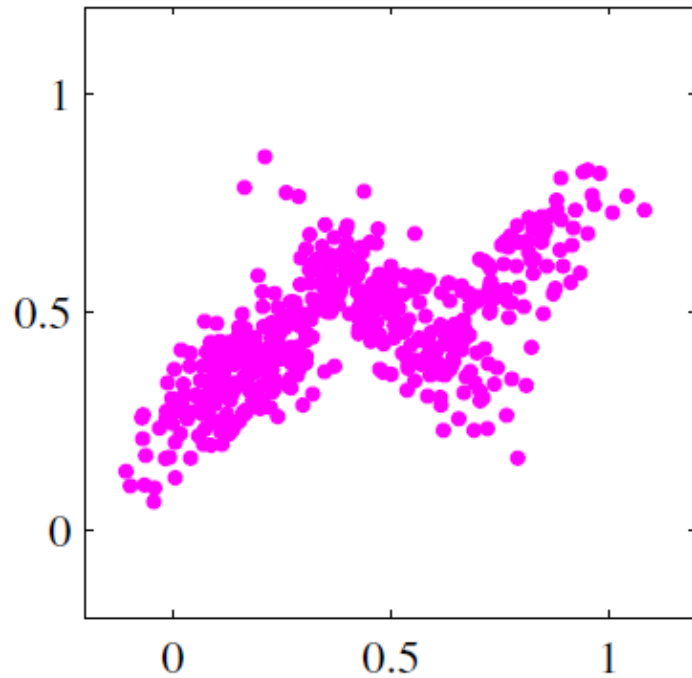
- Generative model: $p(\mathbf{x}) = \sum_z p(\mathbf{x}, z) = \sum_z p(\mathbf{x}|z)p(z)$
- We need to approximate this distribution with a mixture of Gaussians
- A discrete random variable points towards the cluster (latent variable z) and the corresponding cluster is Gaussian distributed.
- It is more flexible than k-Means as it assigns probabilities instead of hard assignment of clusters.
- Uses **Expectation-Maximization Algorithm** to train.

GMM - example



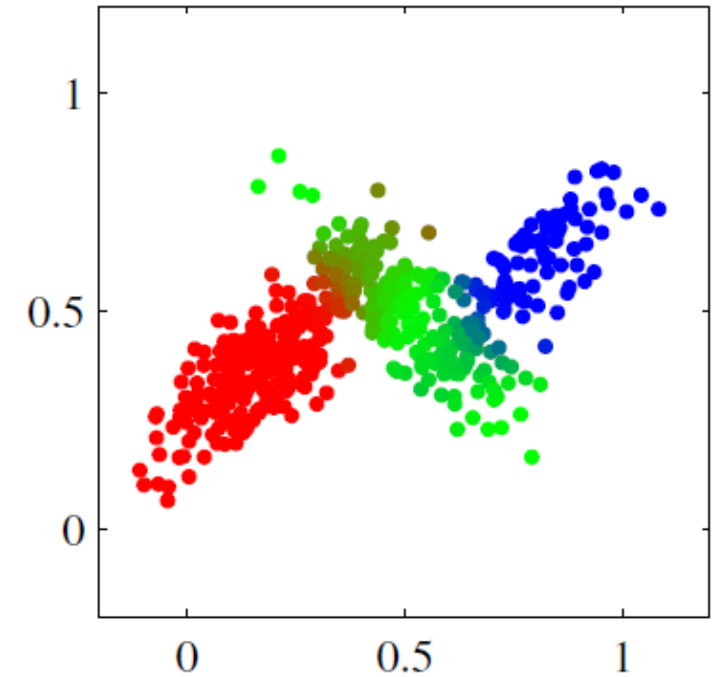
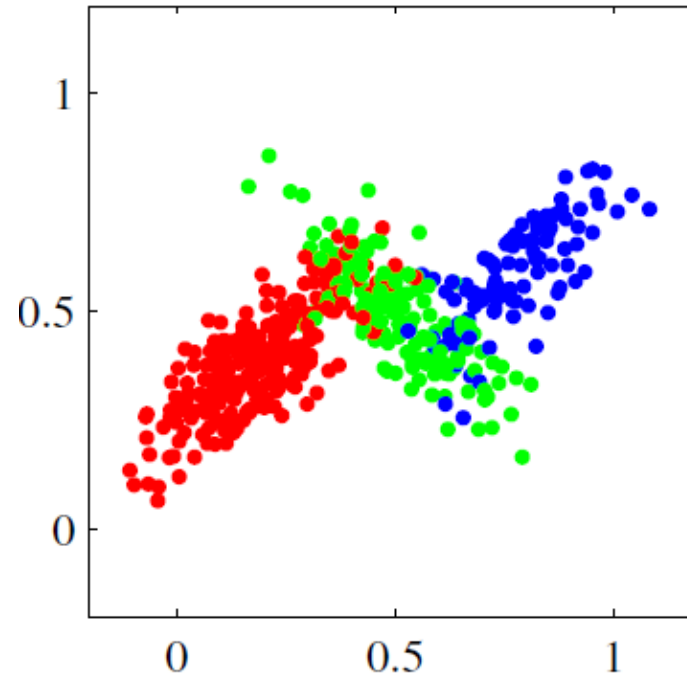
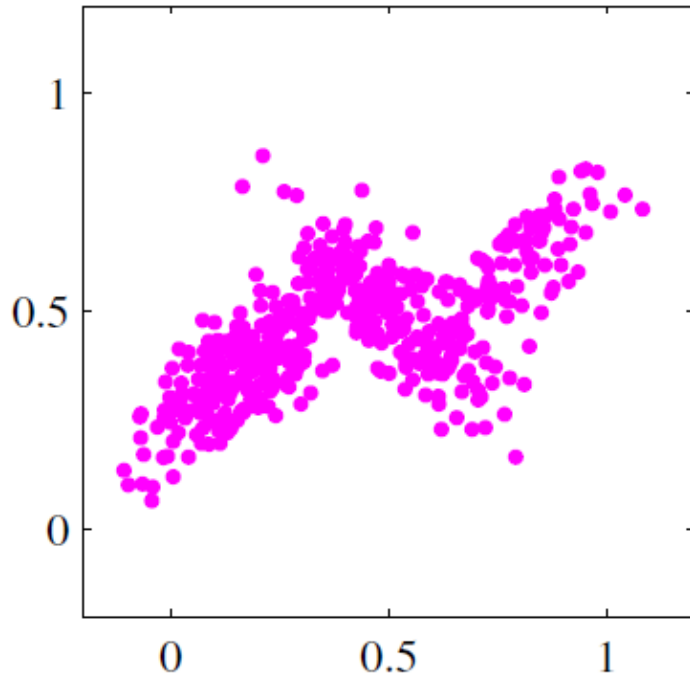
Bishop (slightly changed)

GMM - example



Bishop (slightly changed)

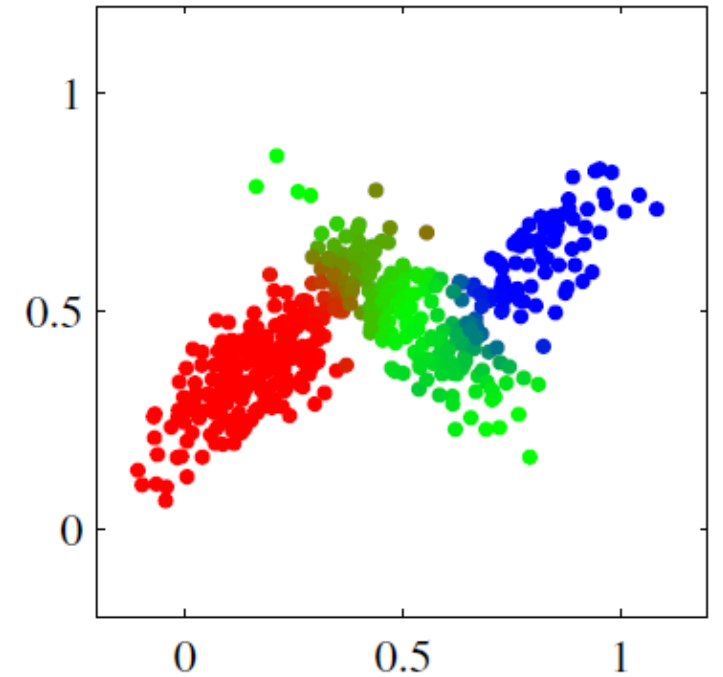
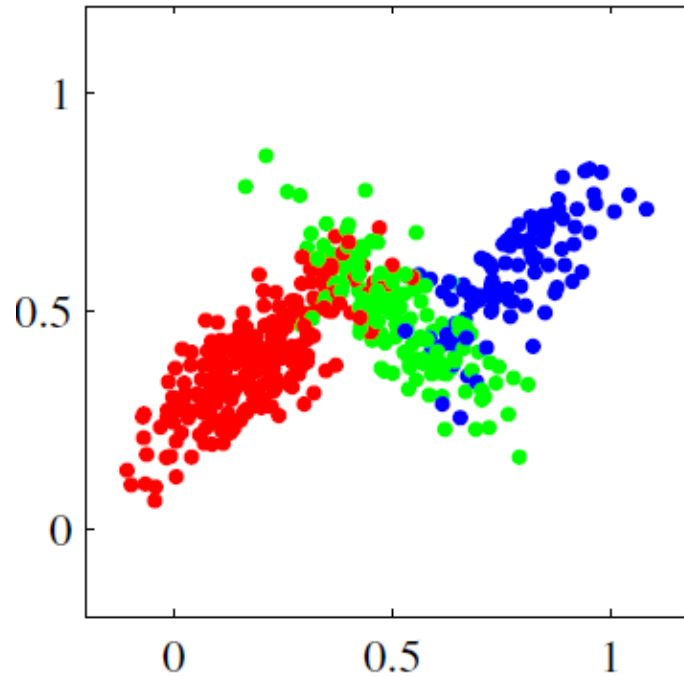
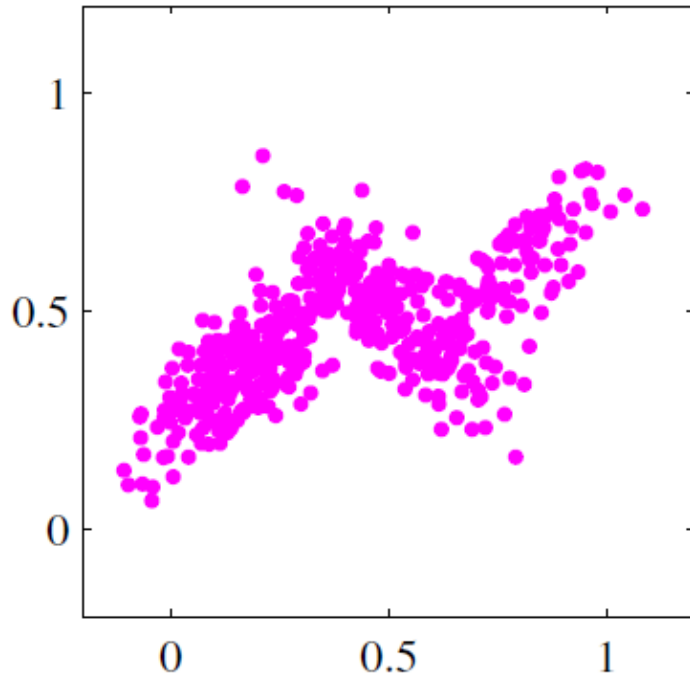
GMM - example



Bishop (slightly changed)

GMM - example

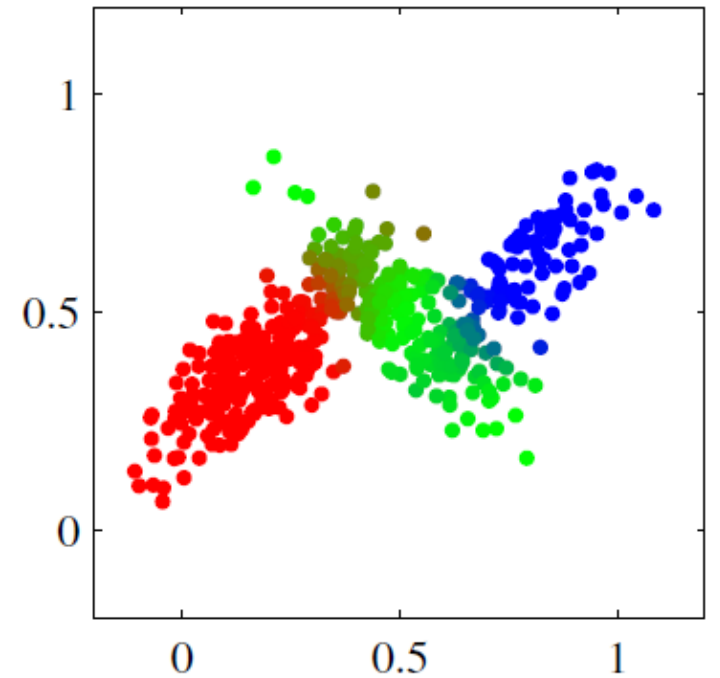
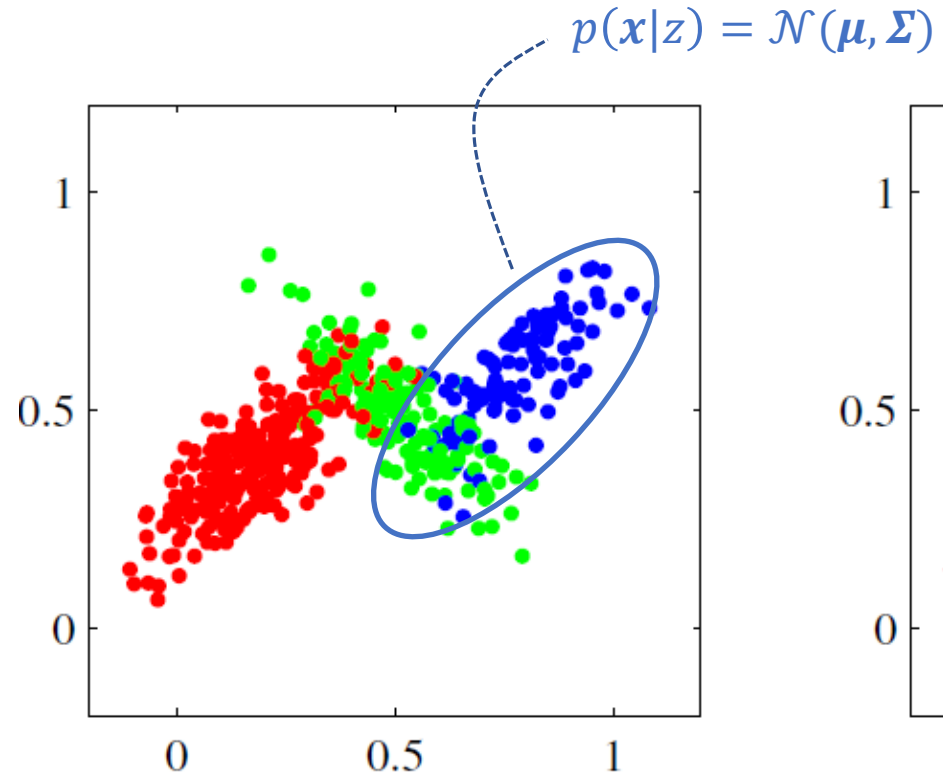
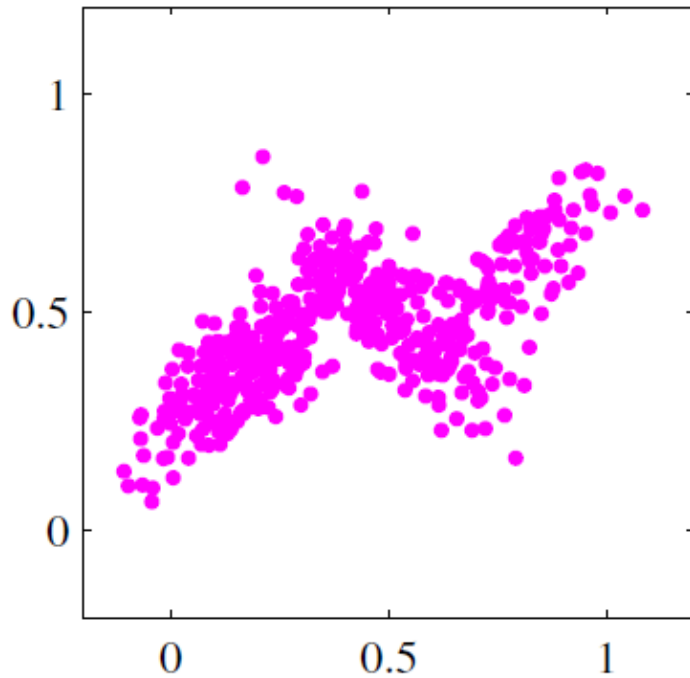
$$p(\mathbf{x}) = \sum_z p(\mathbf{x}, z) = \sum_z p(\mathbf{x}|z)p(z)$$



Bishop (slightly changed)

GMM - example

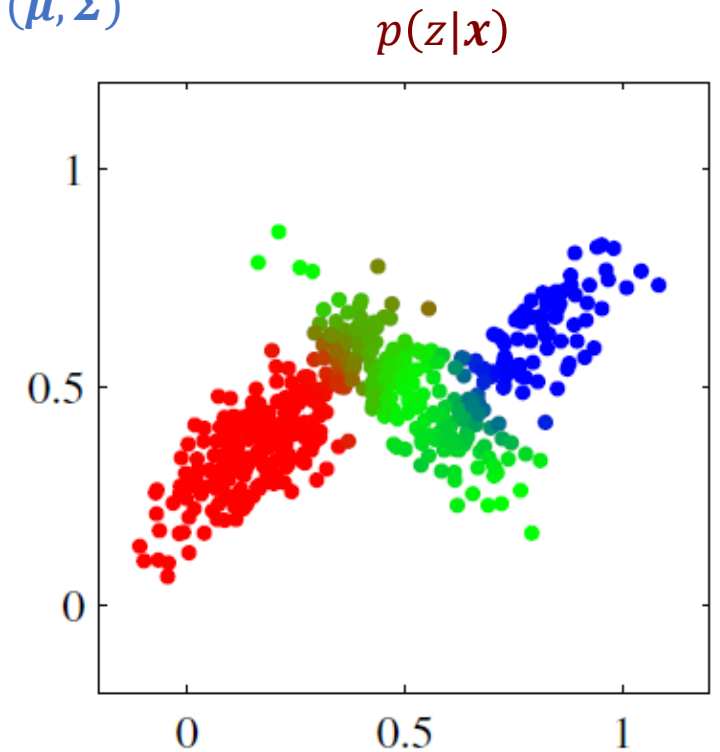
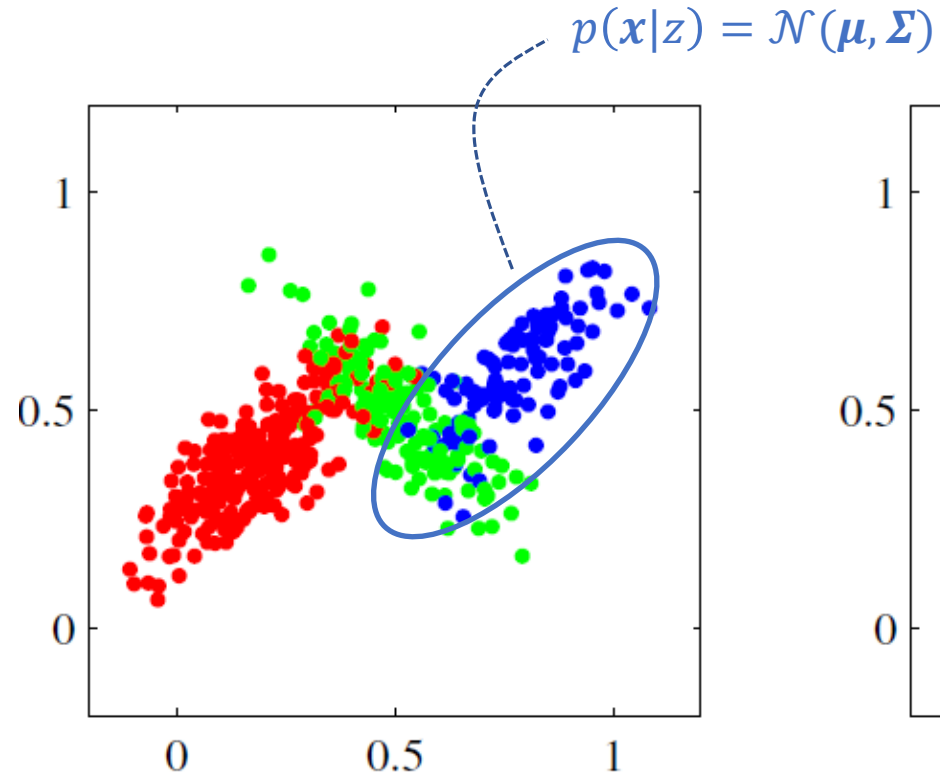
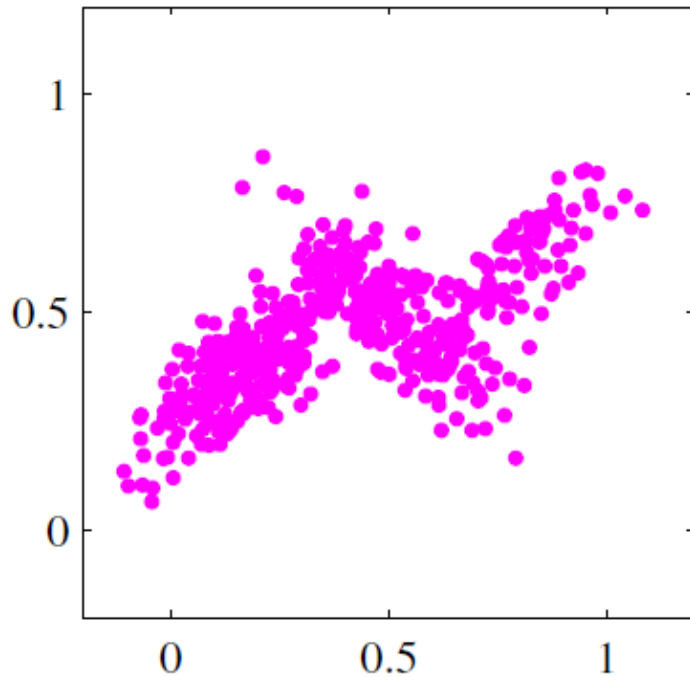
$$p(\mathbf{x}) = \sum_z p(\mathbf{x}, z) = \sum_z p(\mathbf{x}|z)p(z)$$



Bishop (slightly changed)

GMM - example

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



Bishop (slightly changed)

In particular...

- I have data: $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}, \mathbf{x}_n \in \mathbb{R}^D$
- My goal is to find clusters of data by **maximizing the likelihood** of the probabilistic model:

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

where $p(\mathbf{x}|z)$ are **Gaussian distributed** and $p(z)$ is a **generalized Bernoulli** distribution:

$p(z)$ represents the probability of the “appearance” of a cluster.

Modeling of the problem

- $z_k \in \{0,1\}$: one-hot encoding, $k = 1, \dots, K$ (clusters)
 - $p(z_k = 1) = \pi_k, \sum_{k=1}^K \pi_k = 1$
- We use Gaussian distributions to model the clusters. Each cluster has different parameters:

$$p(\mathbf{x}|z_k = 1) =$$

- The joint distribution is

$$p(\mathbf{x}, z_k = 1) =$$

- *The full generative model is:*

$$p(\mathbf{x}) =$$

Modeling of the problem

- $z_k \in \{0,1\}$: one-hot encoding, $k = 1, \dots, K$ (clusters)
 - $p(z_k = 1) = \pi_k, \sum_{k=1}^K \pi_k = 1$
- We use Gaussian distributions to model the clusters. Each cluster has different parameters:

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

- The joint distribution is

$$p(\mathbf{x}, z_k = 1) =$$

- *The full generative model is:*

$$p(\mathbf{x}) =$$

Modeling of the problem

- $z_k \in \{0,1\}$: one-hot encoding, $k = 1, \dots, K$ (clusters)
 - $p(z_k = 1) = \pi_k, \sum_{k=1}^K \pi_k = 1$
- We use Gaussian distributions to model the clusters. Each cluster has different parameters:

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

- The joint distribution is

$$p(\mathbf{x}, z_k = 1) = p(\mathbf{x}|z_k = 1)p(z_k = 1) =$$

- *The full generative model is:*

$$p(\mathbf{x}) =$$

Modeling of the problem

- $z_k \in \{0,1\}$: one-hot encoding, $k = 1, \dots, K$ (clusters)
 - $p(z_k = 1) = \pi_k, \sum_{k=1}^K \pi_k = 1$
- We use Gaussian distributions to model the clusters. Each cluster has different parameters:

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

- The joint distribution is

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

- *The full generative model is:*

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}) = \sum_{\mathbf{z}} p(\mathbf{x}|\mathbf{z})p(\mathbf{z}) = \sum_k \pi_k \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

Bayes...

- Using Bayes we can estimate the a posteriori probability of z given a sample point \mathbf{x} .
- Posterior:

$$\begin{aligned} p(z_k = 1 | \mathbf{x}) &= \frac{p(\mathbf{x} | z_k = 1) p(z_k = 1)}{p(\mathbf{x})} = \frac{p(\mathbf{x} | z_k = 1) p(z_k = 1)}{\sum_{i=1}^K p(\mathbf{x} | z_i = 1) p(z_i = 1)} = \\ &= \frac{\pi_k \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{i=1}^K \pi_i \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)} = \gamma_k \end{aligned}$$

γ_k represents the “responsibility” that cluster k takes for explaining data point \mathbf{x}

Log-likelihood

- Given data: $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}, \mathbf{x}_n \in \mathbb{R}^D$
- I can estimate the log-likelihood of the probabilistic model:

$$\begin{aligned}\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) &= \ln \prod_{n=1}^N p(\mathbf{x}_n|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) \\ &= \sum_{n=1}^N \ln p(\mathbf{x}_n|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) \\ &= \sum_{n=1}^N \ln \sum_k \pi_k \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\end{aligned}$$

- I have the sum inside of the $\ln()$ function, thus, I cannot simplify more.
- How should I maximize the above log-likelihood

Expectation-Maximization Algorithm

- We need to maximize the log-likelihood w.r.t. $\pi_k, \mu_k, \Sigma_k \forall k = 1, \dots, K$
- This is a non-convex problem.
- If we proceed with the derivative of log-likelihood we will end up with, e.g., a μ_k that is a function of γ_k which is what we are looking for (for assignment). In fact, stationary points of the log-likelihood depend on posterior γ_k .
- We can find local minima via an iterative algorithm:
 - update **expected** posteriors γ_k and
 - **maximize** for π_k, μ_k, Σ_k
- This iterative algorithm is called Expectation-Maximization (EM) Algorithm

Expectation-Maximization Algorithm

- Thus, in order to maximize $\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ w.r.t $\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, k = 1, \dots, K$:
 - We solve with γ_k fixed and solve w.r.t. $\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$ (find respective derivatives) of:

$$\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^N \ln \sum_k \pi_k \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

- A useful fact about multivariate Gaussian:

$$\frac{\partial}{\partial \boldsymbol{\mu}_k} \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)(\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1}$$

Expectation-Maximization Algorithm

- After computing all derivatives w.r.t. $\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$ and set equal to zero we get:

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

where $N_k = \sum_{n=1}^N \gamma_{nk}$ ('effective number of points in a cluster')

Expectation-Maximization Algorithm

- After computing all derivatives w.r.t. $\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$ and set equal to zero we get:

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

where $N_k = \sum_{n=1}^N \gamma_{nk}$ ('effective number of points in a cluster')

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n=1}^{N_k} \gamma_{nk} \mathbf{x}_n$$

and

Expectation-Maximization Algorithm

- After computing all derivatives w.r.t. $\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$ and set equal to zero we get:

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

where $N_k = \sum_{n=1}^N \gamma_{nk}$ ('effective number of points in a cluster')

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n=1}^{N_k} \gamma_{nk} \mathbf{x}_n$$

and

$$\boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_{n=1}^{N_k} \gamma_{nk} (\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^T$$

- All solutions depend on the posterior.

Expectation Maximization Algorithm

- Initialize with random $\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$

- Do until convergence:

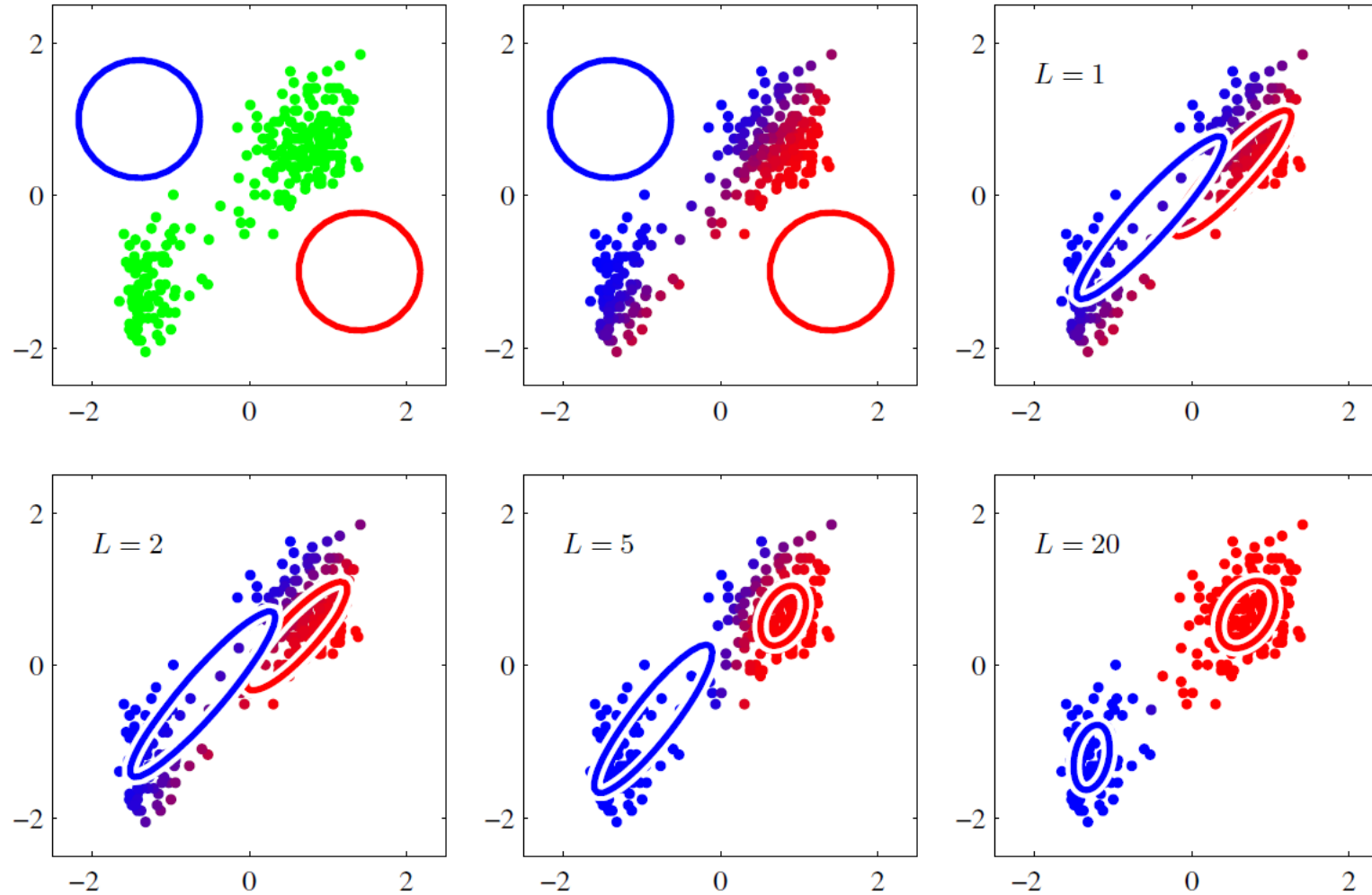
- Update the posterior: $\gamma_{nk} = \frac{\pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{i=1}^K \pi_i \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}$ (expectation step)

- Update the parameters:

$$\pi_k = \frac{N_k}{N}, \boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n=1}^N \gamma_{nk} \mathbf{x}_n \quad (\text{maximization step})$$

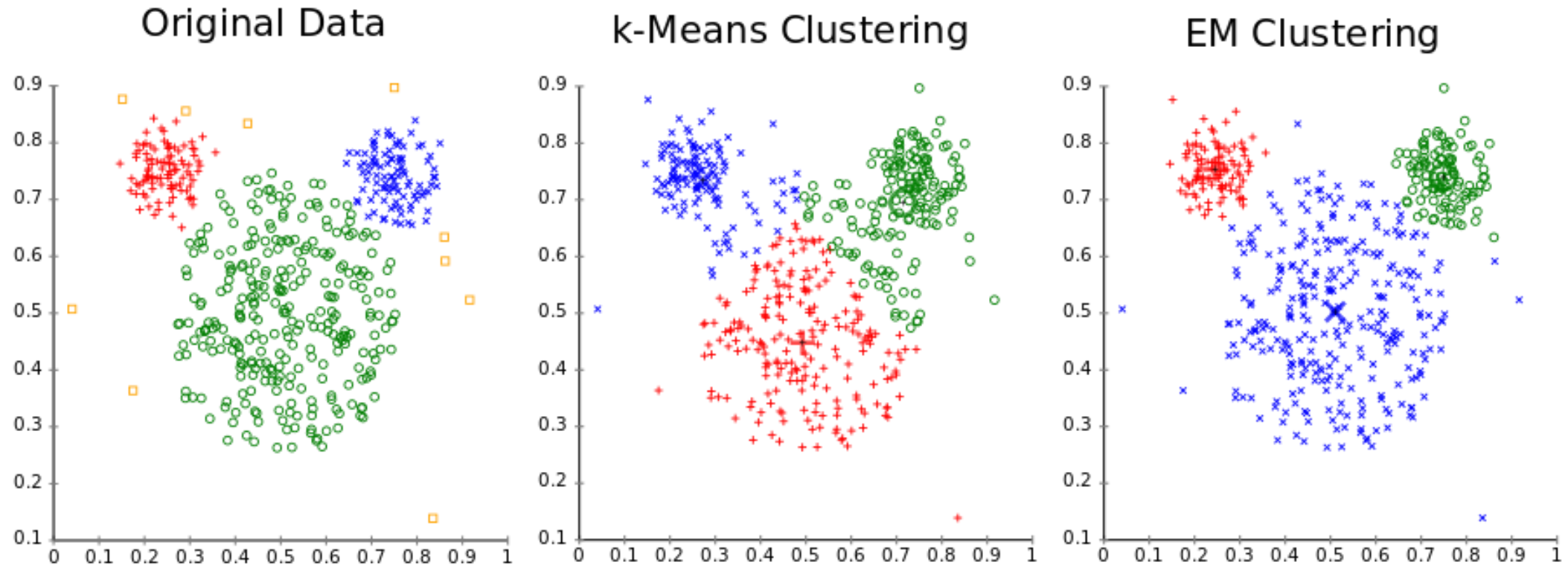
$$\boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_{n=1}^N \gamma_{nk} (\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^T$$

Example



Bishop (slightly changed)

K-Means vs. GMM



wikipedia

How to assign points to clusters

- Soft clusters
 - The posterior γ_{nk} represents the probability that a certain point belongs to a cluster.

$$\gamma_{nk} = p(z_k = 1 | \mathbf{x}_n)$$

- If you need hard clusters:
 - Choose the most likely cluster by:

$$k = \operatorname{argmax}_k \gamma_{nk}$$

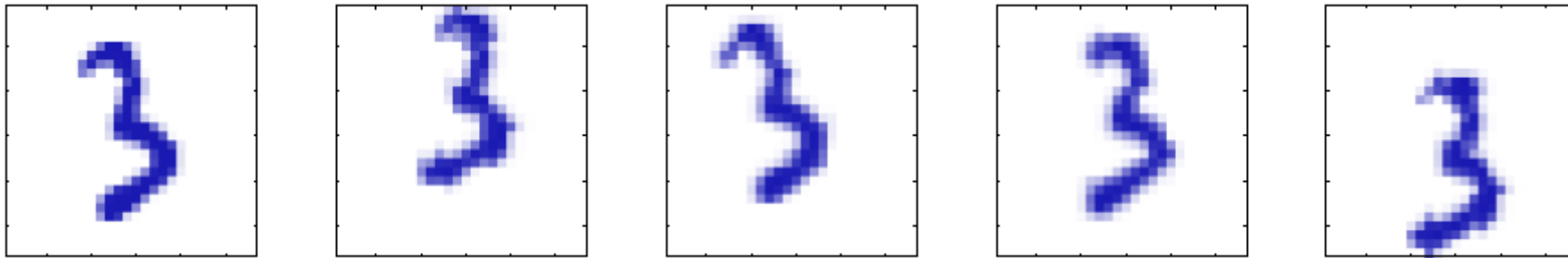
Sum Up

- GMM gives soft-assignment in contrast with k-Means which gives hard assignment.
 - In GMM we can have hard assignment by finding the maximum γ_{nk} across clusters for each data sample.
- GMM is more flexible as we can easily model different parameters per cluster, e.g., covariance matrix.
- On the other hand the **GMM algorithm is slower than k-Means**. Thus, we can use k-Means to initialize the means in the GMM algorithm!
- There is also a chance of **local convergence for GMM algorithm** as well.
- **GMM is similar to Quadratic Discriminant Analysis**. The fact that we have no labels led us to use EM algorithm for training.

Dimensionality Reduction

Dimensionality Reduction - Goal

- We need to model our data using a low dimensional manifold (latent space)
- Example: from one 100×100 image create a new dataset by translation (vertical and horizontal) and rotation.

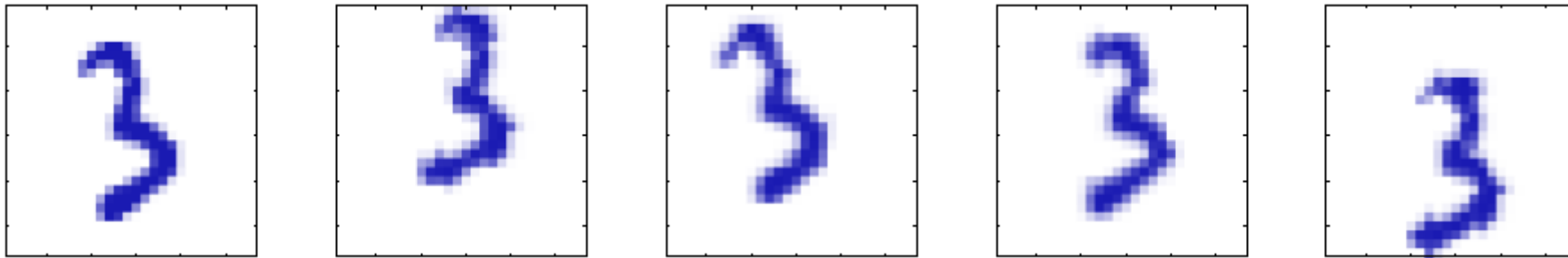


Bishop

- What is the latent space dimension?

Dimensionality Reduction - Goal

- We need to model our data using a low dimensional manifold (latent space)
- Example: from one 100×100 image create a new dataset by translation (vertical and horizontal) and rotation.



Bishop

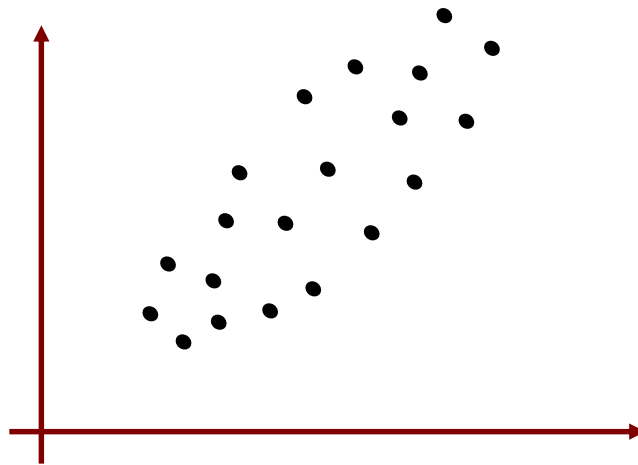
- What is the latent space dimension?
- 3: from 3 latent variables we could generate all 100×100 pixels of the images.

Dimensionality Reduction – Goal (2)

- In a more general example we would have more latent variables, like:
 - Scaling.
 - Hand writing styles
 - different digits
 - etc.
- In any case, the goal is to find a **low dimensional manifold** that describes the data. All these latent variable should be much less than the dimensions of the actual feature space.
- In the previous example, the latent subspace is a **non-linear transformation** of the images.
- In this lecture we will see **Principal Component Analysis** method which refers to **linear latent spaces**.

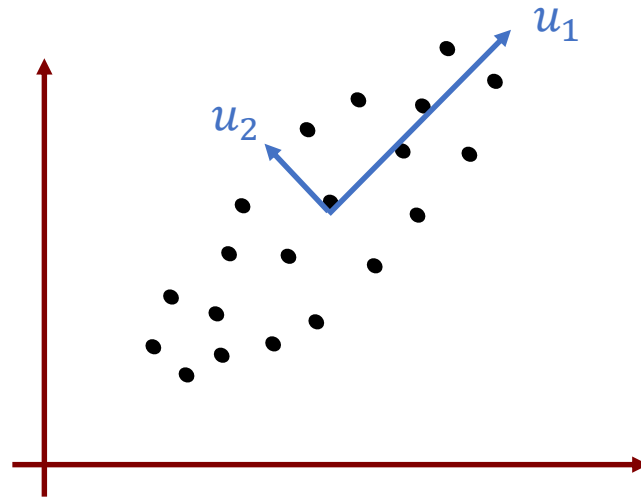
Principal Component Analysis (PCA)

- **Variance:** we seek for a linear projection of our data to a low dimensional space where the variance of our data is maximal.
- PCA projects my data to axes with maximal variation (principal components).



Principal Component Analysis (PCA)

- **Variance:** we seek for a linear projection of our data to a low dimensional space where the variance of our data is maximal.
- PCA projects my data to axes with maximal variation (principal components).



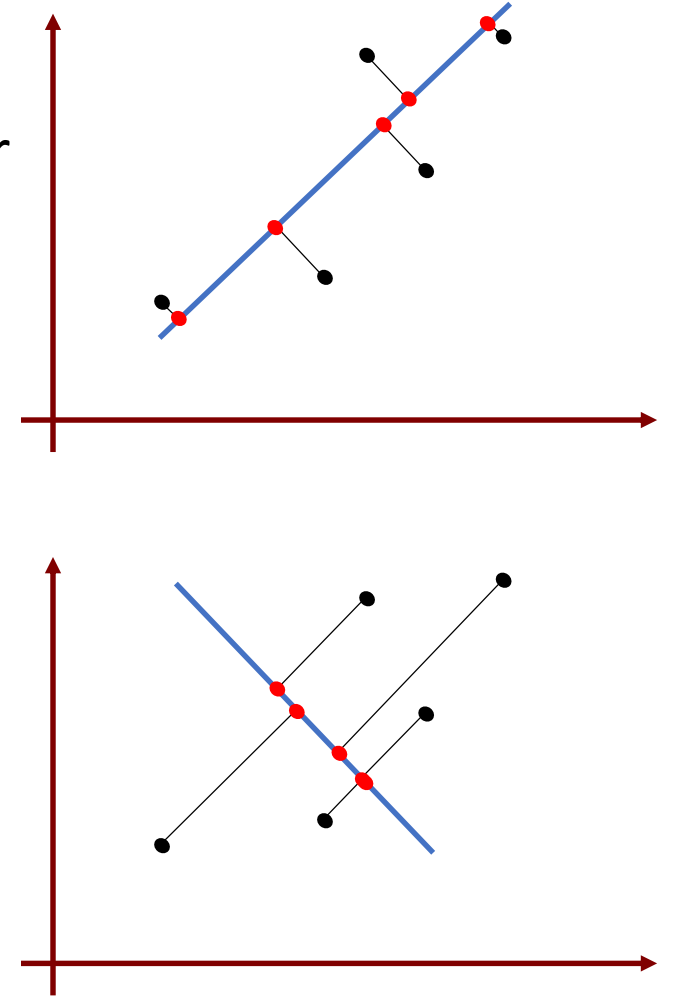
Principal Component Analysis (PCA)

- Data: $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}, \mathbf{x}_n \in \mathbb{R}^D$
- Goal: Project data into $M < D$ dimensional space in order to maximize the **variance** of the projected data
- We need to choose M .
- Recall: mean and covariance of data:

$$\bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n$$

$$\mathbf{C} = \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T$$

where \mathbf{C} is symmetric and positive definite.



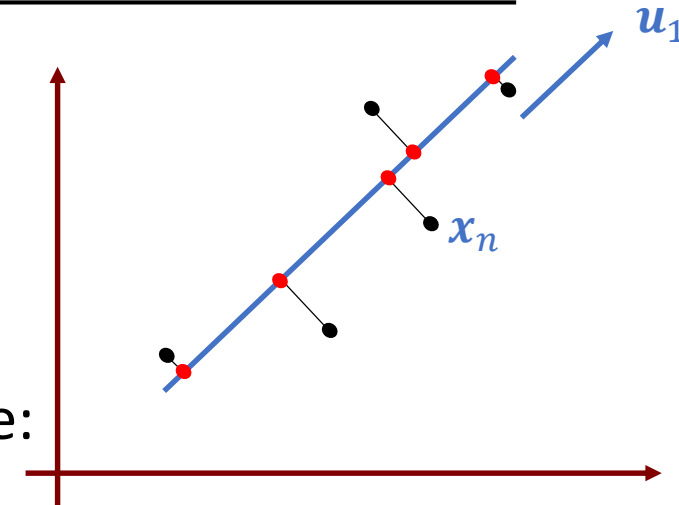
Example – Project in 1D

- We will project our data into the **latent variable** of direction \mathbf{u}_1 : $z_1 = \mathbf{u}_1^T \mathbf{x}_n$ (this is a scalar)
 - the **mean** of the projections is: $\mathbf{u}_1^T \bar{\mathbf{x}} = \mathbf{u}_1^T \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n$
- We are interested only in the **direction** of \mathbf{u}_1 thus we normalize:

$$\|\mathbf{u}_1\|^2 = \mathbf{u}_1^T \mathbf{u}_1 = 1$$

- The **variance** of the projected data is:

$$\begin{aligned} \text{var}[z] &= \frac{1}{N} \sum_{n=1}^N (\mathbf{u}_1^T \mathbf{x}_n - \mathbf{u}_1^T \bar{\mathbf{x}})^2 = \frac{1}{N} \sum_{n=1}^N \left(\mathbf{u}_1^T (\mathbf{x}_n - \bar{\mathbf{x}}) \right)^2 = \\ &= \frac{1}{N} \sum_{n=1}^N \mathbf{u}_1^T (\mathbf{x}_n - \bar{\mathbf{x}}) (\mathbf{x}_n - \bar{\mathbf{x}})^T \mathbf{u}_1 = \mathbf{u}_1^T \left(\frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n - \bar{\mathbf{x}}) (\mathbf{x}_n - \bar{\mathbf{x}})^T \right) \mathbf{u}_1 = \end{aligned}$$



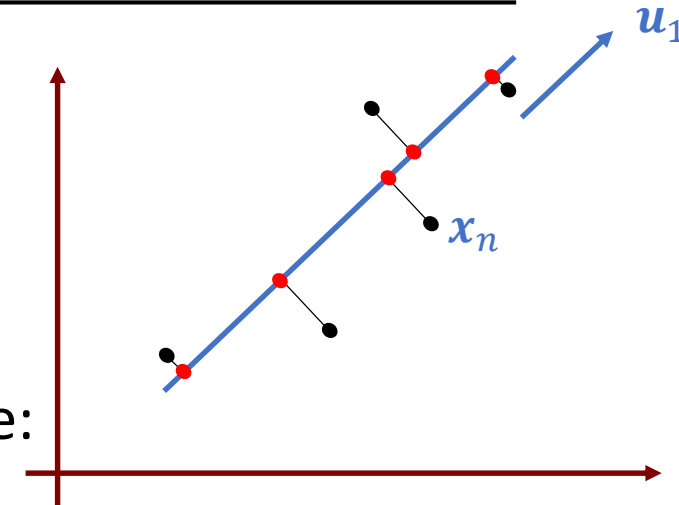
Example – Project in 1D

- We will project our data into the **latent variable** of direction \mathbf{u}_1 : $z_1 = \mathbf{u}_1^T \mathbf{x}_n$ (this is a scalar)
 - the **mean** of the projections is: $\mathbf{u}_1^T \bar{\mathbf{x}} = \mathbf{u}_1^T \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n$
- We are interested only in the **direction** of \mathbf{u}_1 thus we normalize:

$$\|\mathbf{u}_1\|^2 = \mathbf{u}_1^T \mathbf{u}_1 = 1$$

- The **variance** of the projected data is:

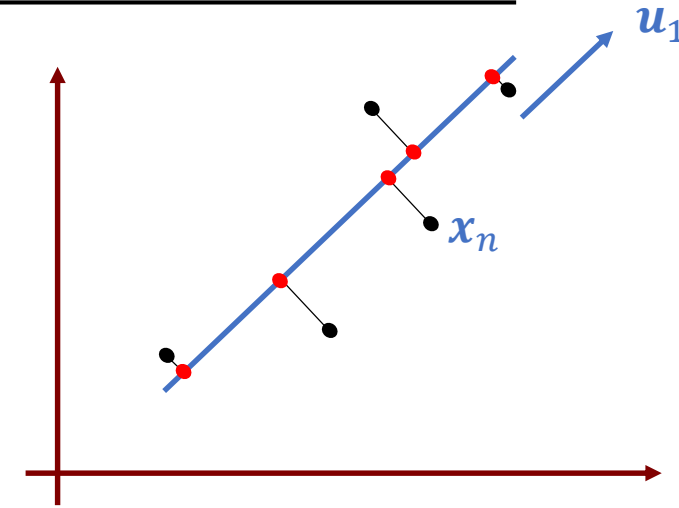
$$\begin{aligned} \text{var}[z] &= \frac{1}{N} \sum_{n=1}^N (\mathbf{u}_1^T \mathbf{x}_n - \mathbf{u}_1^T \bar{\mathbf{x}})^2 = \frac{1}{N} \sum_{n=1}^N \left(\mathbf{u}_1^T (\mathbf{x}_n - \bar{\mathbf{x}}) \right)^2 = \\ &= \frac{1}{N} \sum_{n=1}^N \mathbf{u}_1^T (\mathbf{x}_n - \bar{\mathbf{x}}) (\mathbf{x}_n - \bar{\mathbf{x}})^T \mathbf{u}_1 = \mathbf{u}_1^T \left(\frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n - \bar{\mathbf{x}}) (\mathbf{x}_n - \bar{\mathbf{x}})^T \right) \mathbf{u}_1 = \mathbf{u}_1^T \mathbf{C} \mathbf{u}_1 \end{aligned}$$



Example – Variance Maximization

- We need to maximize the variance in the projected space, thus:

$$\operatorname{argmax}_{\mathbf{u}_1} \mathbf{u}_1^T \mathbf{C} \mathbf{u}_1 \quad s.t. \mathbf{u}_1^T \mathbf{u}_1 = 1$$



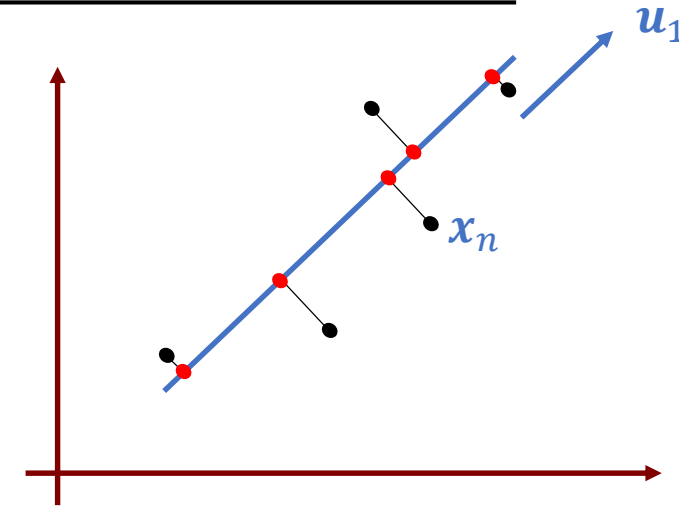
Example – Variance Maximization

- We need to maximize the variance in the projected space, thus:

$$\operatorname{argmax}_{\mathbf{u}_1} \mathbf{u}_1^T C \mathbf{u}_1 \quad s.t. \mathbf{u}_1^T \mathbf{u}_1 = 1$$

- We will use the method of Lagrange Multipliers
- Thus we define the Lagrange function:

$$L(\mathbf{u}_1, \lambda_1) = \mathbf{u}_1^T C \mathbf{u}_1 + \lambda_1 (1 - \mathbf{u}_1^T \mathbf{u}_1)$$



Example – Variance Maximization

$$L(\mathbf{u}_1, \lambda_1) = \mathbf{u}_1^T \mathbf{C} \mathbf{u}_1 + \lambda_1 (1 - \mathbf{u}_1^T \mathbf{u}_1)$$

- Solving for \mathbf{u}_1 we get:

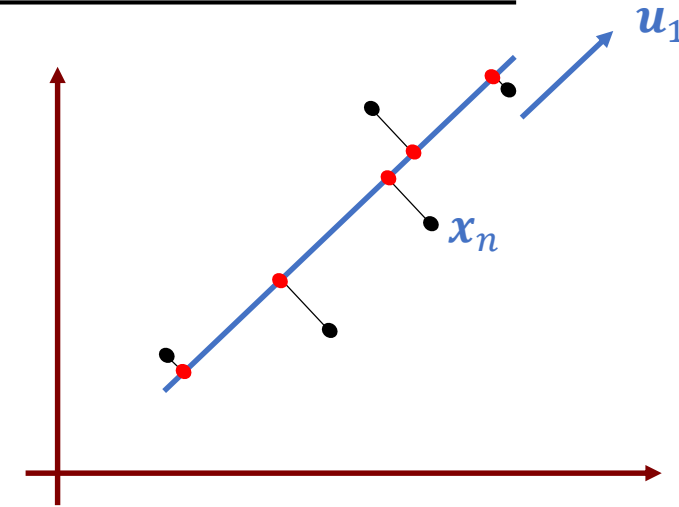
$$\frac{\partial}{\partial \mathbf{u}_1} L(\mathbf{u}_1, \lambda_1) = \mathbf{C} \mathbf{u}_1 - \lambda_1 \mathbf{u}_1 = 0$$

- Thus in essence we need to solve the eigensystem:

$$\mathbf{C} \mathbf{u}_1 = \lambda_1 \mathbf{u}_1$$

where \mathbf{u}_1 and λ_1 are the eigenvector and eigen value of the covariance matrix \mathbf{C}

- What the eigenvalue represents?



Example – Variance Maximization

$$L(\mathbf{u}_1, \lambda_1) = \mathbf{u}_1^T \mathbf{C} \mathbf{u}_1 + \lambda_1 (1 - \mathbf{u}_1^T \mathbf{u}_1)$$

- Solving for \mathbf{u}_1 we get:

$$\frac{\partial}{\partial \mathbf{u}_1} L(\mathbf{u}_1, \lambda_1) = \mathbf{C} \mathbf{u}_1 - \lambda_1 \mathbf{u}_1 = 0$$

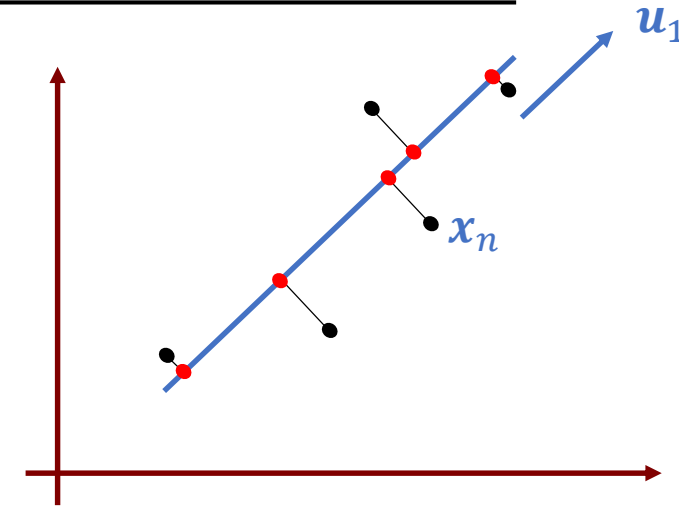
- Thus in essence we need to solve the eigensystem:

$$\mathbf{C} \mathbf{u}_1 = \lambda_1 \mathbf{u}_1$$

where \mathbf{u}_1 and λ_1 are the eigenvector and eigen value of the covariance matrix \mathbf{C}

- What the eigenvalue represents?

$$\mathbf{C} \mathbf{u}_1 = \lambda_1 \mathbf{u}_1 \Rightarrow \mathbf{u}_1^T \mathbf{C} \mathbf{u}_1 = \mathbf{u}_1^T \lambda_1 \mathbf{u}_1 = \lambda_1 \mathbf{u}_1^T \mathbf{u}_1 = \lambda_1$$



PCA – Variance Maximization

- We can repeat the above procedure for M orthogonal vectors
- We will then get the projection:

$$\mathbf{z} = \mathbf{U}_M^T (\mathbf{x}_n - \bar{\mathbf{x}})$$

where $\mathbf{U}_M = [\mathbf{u}_1, \dots, \mathbf{u}_M] \in \mathbb{R}^{D \times M}$

- General PCA procedure:
 - compute $\bar{\mathbf{x}}$ of data
 - find the eigen decomposition of \mathbf{C} where the eigen values are usually arranged as $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_M \geq 0$ (positive semi-definite)
- The total variance of the projected data is: $\sum_{i=1}^M \lambda_i$

PCA – Variance Maximization

- We can repeat the above procedure for M orthogonal vectors
- We will then get the projection:

$$\mathbf{z} = \mathbf{U}_M^T (\mathbf{x}_n - \bar{\mathbf{x}})$$

where $\mathbf{U}_M = [\mathbf{u}_1, \dots, \mathbf{u}_M] \in \mathbb{R}^{D \times M}$

- General PCA procedure:
 - compute $\bar{\mathbf{x}}$ of data
 - find the eigen decomposition of \mathbf{C} where the eigen values are usually arranged as $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_M \geq 0$ (positive semi-definite)
- The total variance of the projected data is: $\sum_{i=1}^M \lambda_i$
- **Intuition:** If I multiply my data with \mathbf{U} , what I actually get is the rotated data where the axons are the \mathbf{u} vectors. My **variables are not correlated** and the covariance matrix in the projected space is **diagonal**!

PCA – Variance Maximization

- We can repeat the above procedure for M orthogonal vectors
- We will then get the projection:

$$\mathbf{z} = \mathbf{U}_M^T (\mathbf{x}_n - \bar{\mathbf{x}})$$

where $\mathbf{U}_M = [\mathbf{u}_1, \dots, \mathbf{u}_M] \in \mathbb{R}^{D \times M}$

- General PCA procedure:
 - compute $\bar{\mathbf{x}}$ of data
 - find the eigen decomposition of \mathbf{C} where the eigen values are usually arranged as $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_M \geq 0$ (positive semi-definite)
- The total variance of the projected data is: $\sum_{i=1}^M \lambda_i$
- **Intuition:** If I multiply my data with \mathbf{U} , what I actually get is the rotated data where the axons are the \mathbf{u} vectors. My variables are not correlated and the covariance matrix in the projected space is diagonal!
- **In practice:** For symmetric positive definite matrices (such as \mathbf{C}) **Singular Value Decomposition** is equivalent to eigen-decomposition and much faster (I need only the M components).

How to choose M?

- We need to preserve as much variance as possible (in the projected space)
- In practice, we usually keep the 90% of the variance of our data in the initial high dimensional space.
- Thus, we need the proportion of the “explained” variance (by the projection) to be above that threshold.
- In essence:

$$\frac{\sum_{i=1}^M \lambda_i}{\sum_{i=1}^D \lambda_i} \geq 0.9$$

Applications

- Dimensionality Reduction
 - Reduce storage space
- Less parameters for classification models (faster implementations)
- Feature extraction
- Data Visualization
- Feature decorrelation

$$\frac{1}{N} \sum_{n=1}^N \mathbf{z}_n \mathbf{z}_n^T = \frac{1}{N} \sum_{n=1}^N \mathbf{U}_M^T (\mathbf{x}_n - \bar{\mathbf{x}}) (\mathbf{x}_n - \bar{\mathbf{x}})^T \mathbf{U}_M = \mathbf{U}_M^T \mathbf{C} \mathbf{U}_M = \mathbf{U}_M^T \mathbf{U} \mathbf{\Lambda} \mathbf{U}^T \mathbf{U}_M = \mathbf{\Lambda}_M$$

- Data whitening: $\mathbf{z} = \mathbf{\Lambda}_M^{-\frac{1}{2}} \mathbf{U}_M^T (\mathbf{x} - \bar{\mathbf{x}})$ (data in the projected space will have unit standard deviation)

PCA - Minimal Reconstruction Error

- We will talk now about the same method (PCA) but **viewed differently**.
- We will now define PCA that minimizes the **reconstruction error**:

$$\frac{1}{N} \sum_{n=1}^N \|\mathbf{x}_n - \tilde{\mathbf{x}}_n\|$$

where $\tilde{\mathbf{x}}_n = U_M \mathbf{z}_n + \bar{\mathbf{x}}$

- Recall: U_M is an unknown orthonormal base.
- Thus, approximate datapoint is generated by a latent variable \mathbf{z} . There are nonlinear methods as well

PCA - Minimal Reconstruction Error

- In the new base we have: $\mathbf{x}_n = \sum_{i=1}^D a_{ni} \mathbf{u}_i$
- We can get the coefficients a_{ni} by simply projecting the vectors onto the orthonormal base:

$$a_{ni} = \mathbf{x}_n^T \mathbf{u}_i$$

- So, $\mathbf{x}_n = \sum_{i=1}^D (\mathbf{x}_n^T \mathbf{u}_i) \mathbf{u}_i$
- For the reconstruction we use only the first M elements of the basis and a shared offset for the remaining dimensions:

$$\tilde{\mathbf{x}}_n = \sum_{i=1}^M (\mathbf{x}_n^T \mathbf{u}_i) \mathbf{u}_i + \sum_{i=M+1}^D b_i \mathbf{u}_i$$

b_i s are shared for all data points.

PCA - Minimal Reconstruction Error

- The difference (reconstruction) error is given by:

$$\mathbf{x}_n - \tilde{\mathbf{x}}_n = \sum_{i=1}^D (\mathbf{x}_n^T \mathbf{u}_i) \mathbf{u}_i - \sum_{i=1}^M (\mathbf{x}_n^T \mathbf{u}_i) \mathbf{u}_i - \sum_{i=M+1}^D b_i \mathbf{u}_i$$

PCA - Minimal Reconstruction Error

- The difference (reconstruction) error is given by:

$$\begin{aligned}\mathbf{x}_n - \tilde{\mathbf{x}}_n &= \sum_{i=1}^D (\mathbf{x}_n^T \mathbf{u}_i) \mathbf{u}_i - \sum_{i=1}^M (\mathbf{x}_n^T \mathbf{u}_i) \mathbf{u}_i - \sum_{i=M+1}^D b_i \mathbf{u}_i \\ &= \sum_{i=M+1}^D (\mathbf{x}_n^T \mathbf{u}_i) \mathbf{u}_i - \sum_{i=M+1}^D b_i \mathbf{u}_i\end{aligned}$$

PCA - Minimal Reconstruction Error

- The difference (reconstruction) error is given by:

$$\begin{aligned}\mathbf{x}_n - \tilde{\mathbf{x}}_n &= \sum_{i=1}^D (\mathbf{x}_n^T \mathbf{u}_i) \mathbf{u}_i - \sum_{i=1}^M (\mathbf{x}_n^T \mathbf{u}_i) \mathbf{u}_i - \sum_{i=M+1}^D b_i \mathbf{u}_i \\ &= \sum_{i=M+1}^D (\mathbf{x}_n^T \mathbf{u}_i) \mathbf{u}_i - \sum_{i=M+1}^D b_i \mathbf{u}_i \\ &= \sum_{i=M+1}^D (\mathbf{x}_n^T \mathbf{u}_i) \mathbf{u}_i - b_i \mathbf{u}_i\end{aligned}$$

- Actually my error is the difference between the shared coefficients and the actual ones for $i \geq M + 1$

PCA - Minimal Reconstruction Error

- We have to minimize the error for both \mathbf{u}_i and b_i :

$$\frac{1}{N} \sum_{n=1}^N \|\mathbf{x}_n - \tilde{\mathbf{x}}_n\|^2 = \frac{1}{N} \sum_{n=1}^N \left\| \sum_{i=M+1}^D (\mathbf{x}_n^T \mathbf{u}_i) \mathbf{u}_i - b_i \mathbf{u}_i \right\|^2$$

- Solution for b_i : $b_i = \bar{\mathbf{x}}^T \mathbf{u}_i$
- We need to solve also for \mathbf{u}_i :

$$\frac{1}{N} \sum_{n=1}^N \left\| \sum_{i=M+1}^D (\mathbf{x}_n^T \mathbf{u}_i) \mathbf{u}_i - (\bar{\mathbf{x}}^T \mathbf{u}_i) \mathbf{u}_i \right\|^2 = \frac{1}{N} \sum_{n=1}^N \left\| \sum_{i=M+1}^D ((\mathbf{x}_n - \bar{\mathbf{x}})^T \mathbf{u}_i) \mathbf{u}_i \right\|^2$$

PCA - Minimal Reconstruction Error

$$\frac{1}{N} \sum_{n=1}^N \left\| \sum_{i=M+1}^D ((\mathbf{x}_n - \bar{\mathbf{x}})^T \mathbf{u}_i) \mathbf{u}_i \right\|^2$$

After numerous computations we can prove that:

$$\frac{1}{N} \sum_{n=1}^N \left\| \sum_{i=M+1}^D ((\mathbf{x}_n - \bar{\mathbf{x}})^T \mathbf{u}_i) \mathbf{u}_i \right\|^2 = \dots = \sum_{i=M+1}^D \mathbf{u}_i^T \mathbf{C} \mathbf{u}_i$$

- Thus in essence we have that:

$$\frac{1}{N} \sum_{n=1}^N \|\mathbf{x}_n - \tilde{\mathbf{x}}_n\|^2 = \sum_{i=M+1}^D \mathbf{u}_i^T \mathbf{C} \mathbf{u}_i =$$

PCA - Minimal Reconstruction Error

$$\frac{1}{N} \sum_{n=1}^N \left\| \sum_{i=M+1}^D ((\mathbf{x}_n - \bar{\mathbf{x}})^T \mathbf{u}_i) \mathbf{u}_i \right\|^2$$

After numerous computations we can prove that:

$$\frac{1}{N} \sum_{n=1}^N \left\| \sum_{i=M+1}^D ((\mathbf{x}_n - \bar{\mathbf{x}})^T \mathbf{u}_i) \mathbf{u}_i \right\|^2 = \dots = \sum_{i=M+1}^D \mathbf{u}_i^T C \mathbf{u}_i$$

- Thus in essence we have that:

$$\frac{1}{N} \sum_{n=1}^N \|\mathbf{x}_n - \tilde{\mathbf{x}}_n\|^2 = \sum_{i=M+1}^D \mathbf{u}_i^T C \mathbf{u}_i = \sum_{i=M+1}^D \lambda_i$$

PCA - Minimal Reconstruction Error

- Thus, I want to minimize:

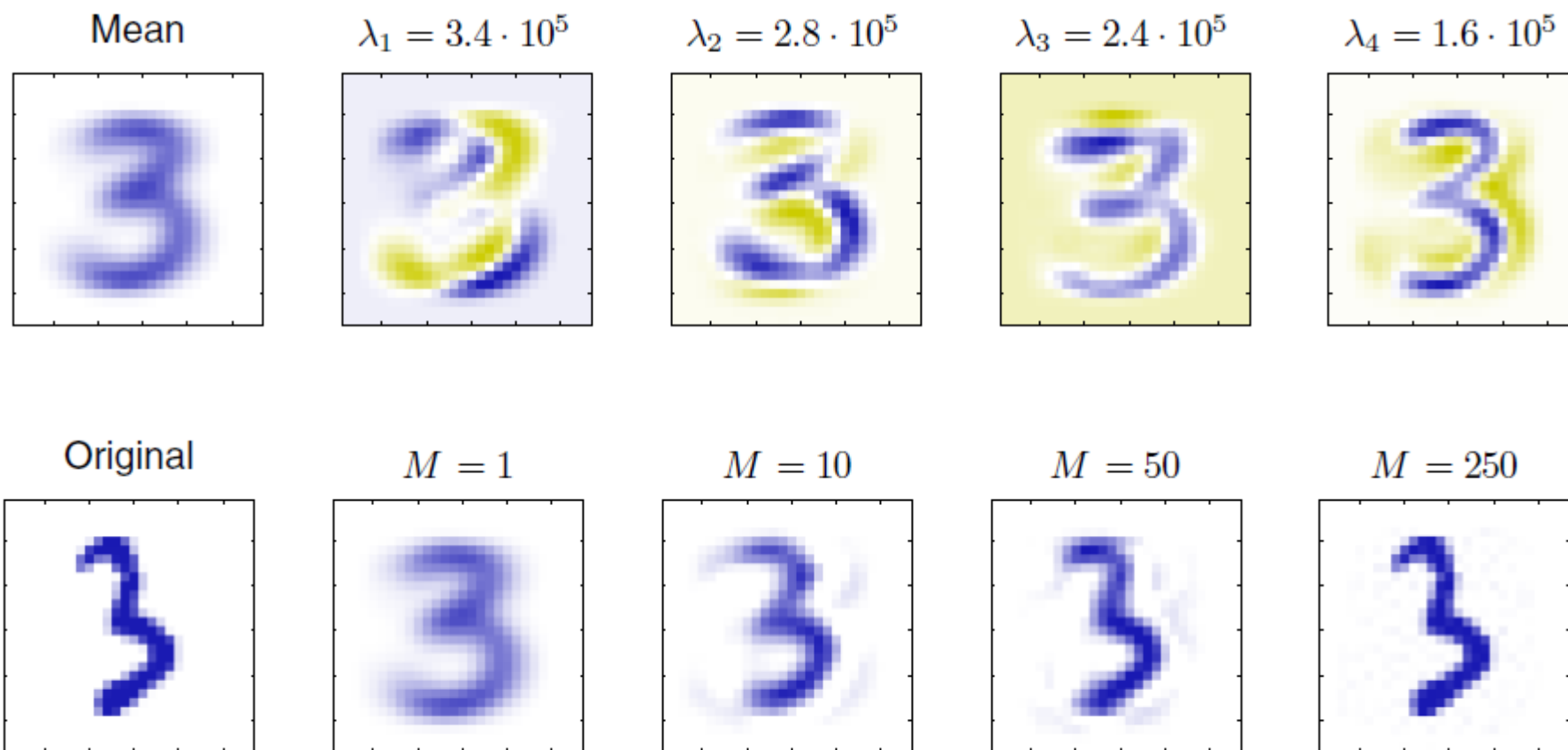
$$\sum_{i=M+1}^D \lambda_i$$

- In essence what I need to do is to find the select the smallest λ_i s for $M + 1 \leq i \leq D$ which is the same as to select the largest λ_i s for $1 \leq i \leq M$ (maximum variance approach)!
- Take-Home message:

minimum reconstruction error \equiv maximum projected variance

Example

$$\tilde{\mathbf{x}}_n = U_M \mathbf{z}_n + \bar{\mathbf{x}}$$



Bishop



ARISTOTLE UNIVERSITY OF THESSALONIKI



FACULTY OF ENGINEERING

Questions?

Pattern Recognition & Machine Learning

Unsupervised learning