

MO433 - Unsupervised Learning

Probability Density Functions (PDFs)

Alexandre Xavier Falcão

Institute of Computing - UNICAMP

afalcao@ic.unicamp.br

Bayes' theorem for classification

Let $\mathbf{x} = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$ be a continuous random variable vector (a **feature vector**) and $y \in \{1, 2, \dots, K\}$ be a discrete random variable of the class label of each sample \mathbf{x} in a dataset.

Bayes' theorem

$$P(y = k|\mathbf{x}) = \frac{p(\mathbf{x}|y = k)P(y = k)}{p(\mathbf{x})}$$

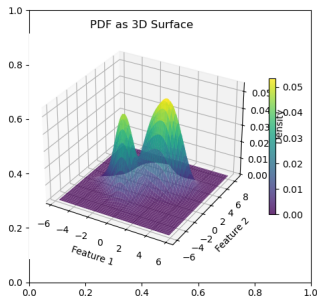
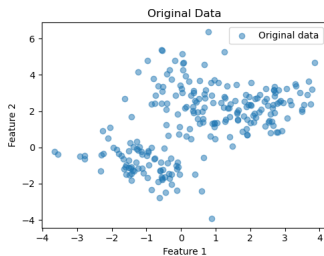
Where:

- ▶ $P(y = k|\mathbf{x})$ is the **posterior probability** of class k given features \mathbf{x} .
- ▶ $p(\mathbf{x}|y = k)$ is the **class-conditional PDF** (likelihood).
- ▶ $P(y = k)$ is the **prior probability** of class k .
- ▶ $p(\mathbf{x}) = \sum_{k=1}^K p(\mathbf{x}|y = k)P(y = k)$ is the **marginal PDF** (evidence).

In this case, the **joint distribution** is $p(\mathbf{x}, y)$.

Geometric interpretation

- ▶ $\mathbf{x} = (x_1, \dots, x_n)^T \in \mathbb{R}^n$ - continuous feature vector.
- ▶ $p(\mathbf{x})$ defines a surface (manifold) in \mathbb{R}^{n+1} .
- ▶ The $(n + 1)$ -th dimension represents density values.



From classification to generation

- ▶ For classification tasks, we want to determine $P(y = k|x)$ - the probability that a feature vector x belongs to class k .
- ▶ $p(x) = \sum_{j=1}^K p(x|y = j)P(y = j)$ is **our main challenge** to estimate.
- ▶ The techniques could also be applied to $p(x|y = k)$, if you use only samples from a class k .
- ▶ Knowledge of $p(x)$ combined with labeled **representative** samples enables semi-supervised classification (pseudo-labeling) of unlabeled data.

We will be interested in using $p(x)$ to **generate** new representative samples.

Agenda

Goal: PDF estimation from **finite data samples**, challenges, and applications.

1. Non-parametric methods - make minimal assumptions about $p(x)$.
2. Parametric methods - Assume $p(x)$ follows a known family (e.g., Gaussian).
3. Manifolds and curse of dimensionality.
4. Applications: KL divergence, model comparison, and a real example.

Non-parametric PDF estimation

For a dataset with N samples, the methods usually rely on a region R of volume V around each sample x .

$$p(x) = \lim_{V \rightarrow 0} \frac{\text{Number of points in region } R \text{ around } x}{N \cdot V}$$

The region R may be:

- ▶ A hypercube of side h (**bandwidth**) and volume $V = h^n$.
- ▶ A hypersphere of radius h (**bandwidth**) and volume

$$V = \frac{\pi^{n/2}}{\Gamma(\frac{n}{2} + 1)} h^n$$

where Γ is Euler's gamma function (**preferred**).

Kernel-based methods

An **exponential kernel** is used around each sample x in both approaches:

$$\phi\left(\frac{x_i - x}{h}\right) = \exp\left(-\frac{\|x_i - x\|^2}{2h^2}\right)$$

The density estimate becomes:

$$p(x) = \frac{1}{N \cdot V} \sum_{i \in S} \phi\left(\frac{x_i - x}{h}\right)$$

where S is the set of samples to consider.

1. **Parzen Window:** $S = \{1, 2, \dots, N\}$ (all samples), V is fixed hypersphere volume (see `code1-parzen_window.py`).
2. **k-NN:** $S = \{k \text{ nearest neighbors}\}$, V adapts to contain exactly k samples (see `code2-knn_density.py`).

Curse of Dimensionality

Let R be a hypersphere with radius $h = 1$, its volume V_n shrinks dramatically as the dimension n increases.

- ▶ $n = 2$: $V_2(1) = \pi \approx 3.14$
- ▶ $n = 5$: $V_5(1) \approx 5.26 \leftarrow$ Peak
- ▶ $n = 10$: $V_{10}(1) \approx 2.55 \leftarrow$ Declining
- ▶ $n = 20$: $V_{20}(1) \approx 0.026 \leftarrow$ Tiny!
- ▶ $n = 100$: $V_{100}(1) \approx 10^{-40} \leftarrow$ Negligible!

Neighbors are **far away** as n increases, making the data points **isolated** and so the density estimates become **unreliable**.

Conversely, to maintain unit volume, radius must grow drastically (see `code3-curse_dimensionality.py`).

Curse of Dimensionality

Let R be a hypersphere with radius $h = 1$, its volume V_n shrinks dramatically as the dimension n increases.

- ▶ $n = 2$: $V_2(1) = \pi \approx 3.14$
- ▶ $n = 5$: $V_5(1) \approx 5.26 \leftarrow$ Peak
- ▶ $n = 10$: $V_{10}(1) \approx 2.55 \leftarrow$ Declining
- ▶ $n = 20$: $V_{20}(1) \approx 0.026 \leftarrow$ Tiny!
- ▶ $n = 100$: $V_{100}(1) \approx 10^{-40} \leftarrow$ Negligible!

Neighbors are **far away** as n increases, making the data points **isolated** and so the density estimates become **unreliable**.

Conversely, to maintain unit volume, radius must grow drastically (see code3-curse_dimensionality.py).

Bottom Line

Parzen window and k-NN methods may fail for $n > 10$ dimensions.

Parametric PDF estimation

Assuming that $p(x)$ belongs to a known parametric family, the **problem** is to estimate its parameters θ from the data samples.

Example: A multivariate Gaussian distribution.

$$p(x) = \mathcal{N}(x|\mu, \Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu)^T \Sigma^{-1}(x - \mu)\right)$$

where $x, \mu \in \mathbb{R}^n$ and $\Sigma \in \mathbb{R}^{n \times n}$ is positive definite (i.e., $x^T \Sigma x > 0$).

Parameters θ :

- ▶ $\mu = (\mu_1, \dots, \mu_n)^T$: mean vector.
- ▶ Σ : $n \times n$ covariance matrix.

Parameter estimation by maximum likelihood

Given N data samples $\{x_1, \dots, x_N\}$, the maximum likelihood estimates

$$\hat{\mu} = \frac{1}{N} \sum_{i=1}^N x_i$$

$$\hat{\Sigma} = \frac{1}{N} \sum_{i=1}^N (x_i - \hat{\mu})(x_i - \hat{\mu})^T$$

Whitening Transform: Data decorrelation

For $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, let

- ▶ \mathbf{Q} be the **eigenvectors** of $\boldsymbol{\Sigma} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^T$, which define the principal axes of the data distribution, and
- ▶ $\boldsymbol{\Lambda} = \text{diag}(\lambda_1, \dots, \lambda_n)$ be its **eigenvalues** that define spread.

Whitening Transform: Data decorrelation

For $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, let

- ▶ \mathbf{Q} be the **eigenvectors** of $\boldsymbol{\Sigma} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^T$, which define the principal axes of the data distribution, and
- ▶ $\boldsymbol{\Lambda} = \text{diag}(\lambda_1, \dots, \lambda_n)$ be its **eigenvalues** that define spread.

The **whitening transform** defines the latent

$$\mathbf{z} = \boldsymbol{\Sigma}^{-1/2}(\mathbf{x} - \boldsymbol{\mu}) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

where

$$\boldsymbol{\Sigma}^{-1/2} = \mathbf{Q}\boldsymbol{\Lambda}^{-1/2}\mathbf{Q}^T$$

$$\boldsymbol{\Lambda}^{-1/2} = \text{diag}(1/\sqrt{\lambda_1}, \dots, 1/\sqrt{\lambda_n})$$

$$p(\mathbf{z}) = \frac{1}{(2\pi)^{n/2}} \exp\left(-\frac{1}{2}\|\mathbf{z}\|^2\right)$$

Applications: Generative models and dimensionality reduction.

See `code4-whitening_transform.py` and `code5-gaussian_analysis.py`.

Other computational simplification

Many generative models (VAEs, normalizing flows) assume **diagonal covariance** for computational efficiency, sometimes applying implicit whitening.

$$\Sigma = \text{diag}(\sigma_1^2, \dots, \sigma_n^2)$$

Simplified Gaussian with independent variables:

$$p(\mathbf{x}) = p(x_1)p(x_2) \dots p(x_n)$$

$$p(\mathbf{x}) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{(x_i - \mu_i)^2}{2\sigma_i^2}\right)$$

Other computational simplification

Many generative models (VAEs, normalizing flows) assume **diagonal covariance** for computational efficiency, sometimes applying implicit whitening.

$$\Sigma = \text{diag}(\sigma_1^2, \dots, \sigma_n^2)$$

Simplified Gaussian with independent variables:

$$p(\mathbf{x}) = p(x_1)p(x_2) \dots p(x_n)$$

$$p(\mathbf{x}) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{(x_i - \mu_i)^2}{2\sigma_i^2}\right)$$

Computational simplification:

This simplification reduces from $\frac{n(n+1)}{2}$ parameters in the full covariance to n parameters in the diagonal one.

Mixture of Gaussians

Real data often exhibits multiple clusters/modes (components).

Solution: Mixture of Gaussians (MoG)

$$p(x) = \sum_{k=1}^K \pi_k \mathcal{N}(x | \mu_k, \Sigma_k)$$

where:

- ▶ K is the number of components.
- ▶ $\pi_k \geq 0$ are mixing weights with $\sum_{k=1}^K \pi_k = 1$.
- ▶ $\mathcal{N}(x | \mu_k, \Sigma_k)$ is the k -th component.

Mixture of Gaussians

Real data often exhibits multiple clusters/modes (components).

Solution: Mixture of Gaussians (MoG)

$$p(x) = \sum_{k=1}^K \pi_k \mathcal{N}(x | \mu_k, \Sigma_k)$$

where:

- ▶ K is the number of components.
- ▶ $\pi_k \geq 0$ are mixing weights with $\sum_{k=1}^K \pi_k = 1$.
- ▶ $\mathcal{N}(x | \mu_k, \Sigma_k)$ is the k -th component.

Parameters to estimate:

$$\theta = \{\pi_1, \dots, \pi_K, \mu_1, \dots, \mu_K, \Sigma_1, \dots, \Sigma_K\}$$

Mixture of Gaussians

Real data often exhibits multiple clusters/modes (components).

Solution: Mixture of Gaussians (MoG)

$$p(x) = \sum_{k=1}^K \pi_k \mathcal{N}(x | \mu_k, \Sigma_k)$$

where:

- ▶ K is the number of components.
- ▶ $\pi_k \geq 0$ are mixing weights with $\sum_{k=1}^K \pi_k = 1$.
- ▶ $\mathcal{N}(x | \mu_k, \Sigma_k)$ is the k -th component.

Parameters to estimate:

$$\theta = \{\pi_1, \dots, \pi_K, \mu_1, \dots, \mu_K, \Sigma_1, \dots, \Sigma_K\}$$

How do we estimate θ ?

KL Divergence: log-likelihood maximization

A PDF defined for the data samples only is

$$p_{data}(x) = \frac{1}{N} \sum_{i=1}^N \delta(x - x_i)$$

where δ is the Kronecker's delta. The KL divergence:

$$\begin{aligned} D_{KL}(p_{data}(x) || p(x; \theta)) &= \int_{\mathbb{R}^n} p_{data}(x) \log \frac{p_{data}(x)}{p(x; \theta)} dx \\ &= \int_{\mathbb{R}^n} p_{data}(x) \log p_{data}(x) dx - \int_{\mathbb{R}^n} p_{data}(x) \log p(x; \theta) dx \\ &= C - \frac{1}{N} \int_{\mathbb{R}^n} \sum_{i=1}^N \delta(x - x_i) \log p(x; \theta) dx = C - \frac{1}{N} \sum_{i=1}^N \log p(x_i; \theta) \end{aligned}$$

where the term $\ell(\theta) = \sum_{i=1}^N \log p(x_i; \theta)$ is the **log-likelihood**.

KL Divergence: log-likelihood maximization

Since $p(\mathbf{x}; \boldsymbol{\theta}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, direct maximum likelihood estimation is intractable due to sum inside logarithm

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

However, minimizing KL divergence is equivalent to maximizing $\ell(\boldsymbol{\theta})$.

$$\min_{\boldsymbol{\theta}} D_{KL}(p_{data}(\mathbf{x}) || p(\mathbf{x}; \boldsymbol{\theta})) \equiv \max_{\boldsymbol{\theta}} \sum_{i=1}^N \log p(\mathbf{x}_i; \boldsymbol{\theta})$$

See `code6-kl_divergence.py`

Expectation-Maximization Algorithm

Key idea: Introduce latent variables $z_{ik} \in \{0, 1\}$

- ▶ $z_{ik} = 1$ if x_i belongs to component k , 0 otherwise
- ▶ Only one $z_{ik} = 1$ per data sample x_i

Expectation-Maximization Algorithm

Key idea: Introduce latent variables $z_{ik} \in \{0, 1\}$

- ▶ $z_{ik} = 1$ if x_i belongs to component k , 0 otherwise
- ▶ Only one $z_{ik} = 1$ per data sample x_i

Complete data log-likelihood

$$\ell_c(\theta) = \sum_{i=1}^N \sum_{k=1}^K z_{ik} [\log \pi_k + \log \mathcal{N}(x_i | \mu_k, \Sigma_k)]$$

Expectation-Maximization Algorithm

Key idea: Introduce latent variables $z_{ik} \in \{0, 1\}$

- ▶ $z_{ik} = 1$ if x_i belongs to component k , 0 otherwise
- ▶ Only one $z_{ik} = 1$ per data sample x_i

Complete data log-likelihood

$$\ell_c(\theta) = \sum_{i=1}^N \sum_{k=1}^K z_{ik} [\log \pi_k + \log \mathcal{N}(x_i | \mu_k, \Sigma_k)]$$

EM Algorithm alternates:

1. **E-step:** Compute $\gamma_{ik}^{(t)} = E[z_{ik} | x_i, \theta^{(t)}] = P(z_{ik} = 1 | x_i, \theta^{(t)})$
2. **M-step:** Maximize $E[\ell_c(\theta^{(t)})]$ w.r.t. $\theta^{(t)}$

Expectation-Maximization Algorithm

Key idea: Introduce latent variables $z_{ik} \in \{0, 1\}$

- ▶ $z_{ik} = 1$ if x_i belongs to component k , 0 otherwise
- ▶ Only one $z_{ik} = 1$ per data sample x_i

Complete data log-likelihood

$$\ell_c(\theta) = \sum_{i=1}^N \sum_{k=1}^K z_{ik} [\log \pi_k + \log \mathcal{N}(x_i | \mu_k, \Sigma_k)]$$

EM Algorithm alternates:

1. **E-step:** Compute $\gamma_{ik}^{(t)} = E[z_{ik} | x_i, \theta^{(t)}] = P(z_{ik} = 1 | x_i, \theta^{(t)})$
2. **M-step:** Maximize $E[\ell_c(\theta^{(t)})]$ w.r.t. $\theta^{(t)}$

Guaranteed: Monotonic increase in likelihood until convergence

EM Algorithm from $\theta^{(0)} = \{\pi_k^{(0)}, \mu_k^{(0)}, \Sigma_k^{(0)}\}$

1: **repeat**

2: **E-step:** For each i, k :

$$\gamma_{ik}^{(t+1)} = \frac{\pi_k^{(t)} \mathcal{N}(x_i | \mu_k^{(t)}, \Sigma_k^{(t)})}{\sum_{j=1}^K \pi_j^{(t)} \mathcal{N}(x_i | \mu_j^{(t)}, \Sigma_j^{(t)})}$$

3: **M-step:** Update parameters:

$$N_k^{(t+1)} = \sum_{i=1}^N \gamma_{ik}^{(t+1)}$$

$$\pi_k^{(t+1)} = \frac{N_k^{(t+1)}}{N}, \quad \mu_k^{(t+1)} = \frac{1}{N_k^{(t+1)}} \sum_{i=1}^N \gamma_{ik}^{(t+1)} x_i$$

$$\Sigma_k^{(t+1)} = \frac{1}{N_k^{(t+1)}} \sum_{i=1}^N \gamma_{ik}^{(t+1)} (x_i - \mu_k^{(t+1)})(x_i - \mu_k^{(t+1)})^T$$

4: $t \leftarrow t + 1$

5: **until** $|\ell_c(\theta^{(t+1)}) - \ell_c(\theta^{(t)})| < \epsilon$.

Model Selection: AIC and BIC

Problem: How many components K should we use in GMM?

- ▶ Too few → **underfitting** (high bias)
- ▶ Too many → **overfitting** (high variance)

Model Selection: AIC and BIC

Problem: How many components K should we use in GMM?

- ▶ Too few \rightarrow **underfitting** (high bias)
- ▶ Too many \rightarrow **overfitting** (high variance)

Solution: Information criteria balance fit vs. complexity

Akaike Information Criterion (AIC)

$$\text{AIC}(K) = -2\ell(\boldsymbol{\theta}) + 2r$$

Bayesian Information Criterion (BIC)

$$\text{BIC}(K) = -2\ell(\boldsymbol{\theta}) + r \log N$$

where r is the number of parameters.

Model Selection: AIC and BIC

Problem: How many components K should we use in GMM?

- ▶ Too few \rightarrow **underfitting** (high bias)
- ▶ Too many \rightarrow **overfitting** (high variance)

Solution: Information criteria balance fit vs. complexity

Akaike Information Criterion (AIC)

$$\text{AIC}(K) = -2\ell(\boldsymbol{\theta}) + 2r$$

Bayesian Information Criterion (BIC)

$$\text{BIC}(K) = -2\ell(\boldsymbol{\theta}) + r \log N$$

where r is the number of parameters.

Strategy: Choose K that **minimizes** AIC or BIC
(code7-em_algorithm.py).

Real-world example

Let a store dataset with regular, premium, and occasional customers to study their behavior.

Customer vectors $x = (x_1, x_2) \in \mathbb{R}^2$

- ▶ x_1 : Monthly spending (\$).
- ▶ x_2 : Visit frequency (visits/month).

File: `code8-customer_analysis.py`

Preview of the next lecture

Real-world data often lives in high-dimensional spaces ($n \gg 10$) where:

- ▶ Direct visualization is impossible.
- ▶ Traditional density estimation fails.
- ▶ We need **dimensionality reduction** techniques.

Preview of the next lecture

Real-world data often lives in high-dimensional spaces ($n \gg 10$) where:

- ▶ Direct visualization is impossible.
- ▶ Traditional density estimation fails.
- ▶ We need **dimensionality reduction** techniques.

Fundamental questions

- ▶ How can we visualize PDFs in \mathbb{R}^{100+} ?
- ▶ What is the intrinsic dimensionality of real data?
- ▶ How do we preserve PDF structure when reducing dimension?