

# Intelligent Systems for Pattern Recognition

Federico Matteoni

A.A. 2021/22

# Index

0.1	Introduction . . . . .	2
0.2	Pattern Recognition . . . . .	2
0.2.1	Signals . . . . .	2
0.2.2	Image Processing . . . . .	5

## 0.1 Introduction

Prof.s: Davide Bacciu and Antonio Carta

**Objectives** Train ML specialists capable of: designing novel learning models, developing pattern recognition applications using ML, developing intelligent agents using **Reinforcement Learning**.

We're referring to images and signals, but not limited to that: practical applications.

Focusing on challenging and complex data: **machine vision** (noisy, hard to interpret, semantically rich...) and **structured data** (relational information: sequences, trees, graphs...)

Natural Language Processing will be used as an example, but will not be the focus of this course.

**Methodology-Oriented Outcomes** Gain in-depth knowledge of advanced machine learning models, understanding the underlying theory. This gives the ability to read and understand and discuss research works in the field.

**Application-Oriented Outcomes** Learn to address modern pattern recognition applications, gain knowledge of ML, PR and RL libraries and be able to develop an application using ML and RL models.

**Prerequisites** Knowledge of ML fundamentals, mathematical tools for ML and Python.

## 0.2 Pattern Recognition

Automated recognition of meaningful patterns in noisy data.

### Origins

**Viola-Jones Algorithm** Framework for face recognition. Sum pixel in white area and subtract those in the black portion. The VJ algorithm positions the masks on the image and combines the responses (training set of  $\simeq 5k$  images with hand-aligned filters)

#### An historical view

1. Identification of distinguishing features of the object/entity (**feature detection**)
2. Extraction of features for the defining attributes (**feature extraction**)
3. Comparison with known patterns (**matching**)

Basically, lots of time spent hand-engineering the best data features.

**A modern view** Data is thrown into a neural network. A single stage process with a data crushing-and-munching neural network spitting out prediction, which encapsulates the three historical steps. But the time is now spent in fine-tuning the neural network.

**The deep learning Lego** Creating applications by putting together various combinations of CNN and LSTM modules.

### 0.2.1 Signals

Signals are time series: a sequence of measurements in time. Examples of sources are: medicine, finance, geology, IoT, biometrics...

**Formalization** A time series  $x$  is a sequence of measurements in time  $t$

$$x = x_0, \dots, x_N$$

where  $x_t$  or  $x(t)$  is the measurement at time  $t$ .

Observation can be at **irregular** time intervals.

We assume **weakly stationary** (or second-order stationary) data

$$\forall t \ E[x_t] = \mu$$

$$\forall t \ \text{Cov}(x_{t+\tau}, x_t) = \gamma_\tau \text{ with } \gamma \text{ depending only on the lag } \tau$$

## Goals

### Description

**Analysis:** identify and describe dependencies in data

**Prediction:** forecast next values given information up to  $t$

**Control:** adjust parameters of the generative process to make the time series fit a target

## Key Methods

**Time domain analysis:** assesses how a signal changes over time (correlation, convolution, auto-regressive models)

**Spectral domain analysis:** assesses the distribution of the signal over a range of frequencies (Fourier analysis, wavelets)

## Time Domain Analysis

### Mean

$$\hat{\mu} = \frac{1}{N} \sum_{t=1}^N x_t$$

Can be used to subtract mean from values and "standardize" the two series.

**Autocovariance** For lag  $-N \leq \tau \leq N$

$$\hat{\gamma}_x(\tau) = \frac{1}{N} \sum_{t=1}^{N-|\tau|} (x_{t+|\tau|} - \hat{\mu})(x_t - \hat{\mu})$$

**Autocorrelation** The correlation of a signal with itself.

$$\hat{\rho}_x(\tau) = \frac{\hat{\gamma}_x(\tau)}{\hat{\gamma}_x(0)}$$

We can compute this with every possible  $\tau$ , finding the max/min which gives the  $\tau$  where the autocorrelation is max/min, which means the lag where the signal starts repeating itself. The lags near zero typically dominates, so we want the maximum lag reasonably far from 0.

### Autocorrelation plot

**Cross-Correlation** A measure of similarity of  $x_1$  and  $x_2$  as a function of a time lag  $\tau$

$$\phi_{x_1 x_2}(\tau) = \sum_{t=\max\{0, \tau\}}^{\min\{(T_1-1+\tau), (T_2-1)\}} x_1(t - \tau) \cdot x_2(t)$$

**Normalized cross-correlation** Returns an amplitude independent value

$$\bar{\phi}_{x_1 x_2}(\tau) = \frac{\phi_{x_1 x_2}(\tau)}{\text{todo}} \in [-1, +1]$$

With  $\bar{\phi}_{x_1 x_2}(\tau) = +1$  mean that the two time series have the exact same shape if aligned at time  $\tau$ . Nearing  $-1$  we get the maximum anticorrelation, same shape but opposite sign. Near 0 we get that the two signals are completely **linearly** uncorrelated.

Note that we measure **linear correlation**.

Cross correlation looks like the convolution

$$(f * g)[n] = \sum_{t=-M}^M f(n-t)g(t)$$

but we have a flipped sign ( $n-t$  instead of  $t-\tau$ ).

Cross-correlation is not symmetric, whereas convolution is ( $f * g = g * f$ ).

**Autoregressive Process** A timeseries autoregressive process (AR) of order  $K$  is the linear system

$$x_t = \sum_{k=1}^K \alpha_k x_{t-k} + \epsilon_t$$

Autoregressive means  $x_t$  regresses on itself

$\alpha_k \Rightarrow$  linear coefficients  $|\alpha| < 1$

$\epsilon_t \Rightarrow$  sequence of independent and identically distributed values with mean 0 and fixed variance.

We look backward  $K$  steps, so limited memory.

**ARMA** Autoregressive with Moving Average process

$$x_t = \sum_{k=1}^K \alpha_k x_{t-k} + \sum_{q=1}^Q \beta_q \epsilon_{t-1} + \epsilon_t$$

With  $\epsilon_t$  Random white noise (again)

The current time series values is the result of a regression on its past values plus a term that depends on a combination of stochastically uncorrelated information

**Estimating Autoregressive Models** Need to estimate: the values of the linear coefficients  $\alpha_t$  and  $\beta_t$  and the order of the autoregressor  $K$  and  $Q$

Estimation of the  $\alpha$ ,  $\beta$  is performed with the Levinson-Durbin Recursion (`levinson(x, K)` in matlab, and included in several Python modules).

The order is often estimated with a Bayesian model selection criterion, choosing the largest  $K$  and  $Q$  possible. E.g.: BIC, AIC...

The set of autoregressive parameters  $\alpha_{i,1}, \dots, \alpha_{i,K}$  fitted to a specific time series  $x_i$  is used to confront it with other time series. Same thing for  $\beta$  so we can use  $\alpha$  for both sets.

**Comparing time series by AR**

timeseries clustering

novelty/anomaly detection

**Spectral Domain Analysis**

Analyze the time series in the frequency domain. Key idea: decomposing the time series into a linear combination of sines and cosines with random and uncorrelated coefficients. So a **regression on sinusoids** with Fourier analysis.

**Fourier Transform** Discrete Fourier Transform (DFT): transform a time series from the time domain to the frequency domain. Can be easily inverted back to the time domain.

Useful to handle periodicity in the time series: seasonal trends, cyclic processes...

**Representing functions** We know that, given an orthonormal system for  $E$  we can use linear combinations of the basis  $\{e_1, \dots\}$  to represent any function  $f \in E$

$$\sum_{k=1}^{\infty} \langle f, e_k \rangle e_k$$

Given the orthonormal system

$$\left\{ \frac{1}{\sqrt{2}}, \sin(x), \cos(x), \sin(2x), \cos(2x), \dots \right\}$$

The linear combination above becomes the Fourier series `todo`

**Representing function in Complex space** Using  $\cos(kx) - i \sin(kx) = e^{-ikx}$  with  $i = \sqrt{-1}$  we can rewrite the Fourier series as

$$\sum_{k=-\infty}^{\infty} c_k e^{ikx}$$

on the orthonormal system

$$\{1, e^{ix}, e^{-ix}, e^{2ix}, e^{-2ix}, \dots\}$$

**Representing Discrete Time Series** Consider  $x$  of length  $N$  and  $x_n \in \mathbb{R}$ . Using the exponential formulation, the orthonormal system is finite, from  $e_0$  to  $e_{N-1}$  each  $\in C^N$ . The  $n$ -th component of the  $k$ -th vector is

$$[e_k]n = e^{\frac{-2\pi i n k}{N}}$$

**Discrete Fourier Transform** Given a time series  $x = x_0, \dots, x_{N-1}$  its DFT is the sequence

$$\text{Spectral domain } X_k = \sum_{n=0}^{N-1} x_n e^{\frac{-2\pi i n k}{N}} \quad \text{Time domain}$$

And can be inverted

$$x_k = \frac{1}{N} \sum_{n=0}^{N-1} X_n e^{\frac{2\pi i n k}{N}}$$

### Basic Spectral Quantities in SFT

Amplitude  $A_k = |X_k| = \sqrt{\text{Re}^2(X_k) + \text{Im}^2(X_k)}$

Power  $P_k = \frac{|X_k|^2}{N}$ , more used in reality and under some conditions this is a reasonable estimate of

**DFT in Action** We use the DFT elements  $X_1, \dots, X_K$  as representation of the signal to train the predictor/classifier. This representation can reveal patterns that are not clear in the time domain.

## 0.2.2 Image Processing

Bidimensional series. Basically same approach to signals.

### Descriptors

An image is a matrix of pixel intensities or color values (RGB). There are other representations, not interesting for this course. CIE-LUV often used in image processing due to perceptual linearity (image difference is more coherent)

**Machine Vision Applications** For example region of interest, or object classification.

Even pixel-level tasks, for example image segmentation (regions of the image) or semantic segmentation (classifying regions of the image).

Up one level of abstraction: automated image captioning, requiring identifying objects, generating sentences and ranking those sentences.

### Key Questions

How to represent visual information? It has to be:

Informative, carrying all the information

Invariant to photometric (different illuminations) and geometric transformation (position in the picture, rotation...)

Efficient for indexing and querying

How to identify informative parts?

Whole image is generally not a good idea

Must lead to good representations

**Image Histograms** One of the first answer. Describes the distribution of some visual information on the whole image: colors, edges, corners... depending on the goals.

**Color Histograms**, one of the earliest image descriptors.

Count the number of pixels of a given color (normalize!). We need to discretize and group the RGB colors.

Any information concerning shapes and position is lost. Two image with a random permutation of the same pixels produce the same color histograms.

Images can be compared, indexed and classified based on their color histogram representation.

Can be computed with OpenCV in Python.

**Describing Local Image Properties** We need something less global, on a local level. Capturing information on image regions, extract **multiple local descriptors**: different location, different scale...

Several approaches, typically performing convolution between a filter and the image region. Using filters sensitive to specific features we can extract many kind of information.

## Localized Descriptors

**Intensity Vector** The simplest form of localized descriptor: a vector  $n \cdot m$  of the pixels of a single patch of the image with dimensions  $n, m$ . The vector can be normalized to make it invariant to intensity variations.

But rotating gives a different vector. A more robust representation is an histogram of this vector.

**Distribution-Based Descriptors** Represent local patches by histograms describing properties of the pixels in the patch. The simplest is an histogram of intensity values, but it's not invariant enough even if normalized.

We want a descriptor invariant to illumination (normalization), scale (captured at multiple scale) and geometric transformations (rotation invariant). We want locality, histogram based and invariant to geometric transformation.

## SIFT Scale Invariant Feature Transform

1. Center the image patch on a pixel  $x, y$  of the image  $I$
2. Represent image at scale  $\sigma$  (controls how close to look at the image)

Convolve the image with a Gaussian filter with standard variation  $\sigma$ , basically computing average of pixels with the coefficient taken from a Gaussian distribution. With a smooth Gaussian, we artificially smooth the object, and vice versa. We can compute different versions of the image.

$$L_{\sigma}(x, y) = G(x, y, \sigma) * I(x, y)$$

$$G(x, y, \sigma) = \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

3. Compute the **gradient of intensity** in the patch, extracting magnitude  $m$  and orientation  $\Theta$  using finite differences.

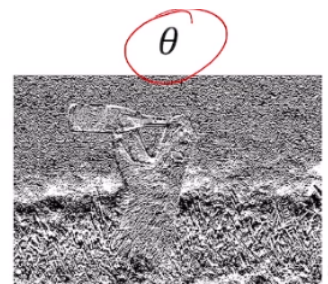
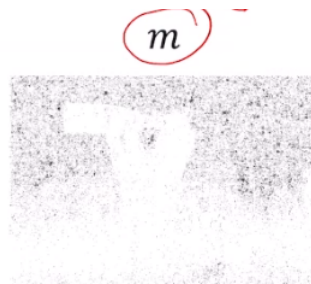
## Gaussian Filter of an Image

```
lscale = imgaussfilt(l, sigma);
```

$\sigma = 5$



$\sigma = 0.05$



#### 4. Create **gradient histogram**

$4 \times 4$  gradient window

Histogram of  $4 \times 4$  per window on 8 orientation bins

Gaussian weighting on center keypoint (width =  $1.5\sigma$ )

$4 \times 4 \times 8 = 128$  descriptor size

