$$D \Rightarrow \begin{array}{c|cccccc} & 0 & 1 & 2 & \dots & 49 \\ \hline 0 & & & & & \\ 1 & & & & & \\ 2 & & & & & \\ 3 & & & & & \end{array}$$

PROBABILITY AND DENSITY ESTIMATION

$$X = \begin{array}{c|c} & 0 \\ \hline 0 & \\ 1 & \\ 2 & \\ 3 & \end{array}$$

PRIMA COLONNA DI X

$(4,1) \xrightarrow{\text{VT.VCOL}} (1,4)$

Laboratory 4

In this laboratory we will focus on computing probability densities and ML estimates.

DEVI CALCOLARLA SU OGNI CLASSE

Multivariate Gaussian density

$$\log \mathcal{N}(x|\mu, C) = -\frac{\mu}{2}\log 2\pi - \frac{1}{2}\log|C| - \frac{1}{2}(x-\mu)^T C^{-1}(x-\mu)$$

SEGNO, VAL = MP.LINALG.SLOGDET

MP.LINALG.INV

The Multivariate Gaussian (MVG) density is defined as

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{\frac{M}{2}}|\boldsymbol{\Sigma}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}$$

where M is the size of the feature vector $\boldsymbol{x}$, and $|\boldsymbol{\Sigma}|$ is the determinant of $\boldsymbol{\Sigma}$.
To avoid numerical issues due to exponentiation of large numbers, in many practical cases it's more convenient to work with the logarithm of the density

$$\log\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = -\frac{\overset{4}{M}}{2}\log 2\pi - \frac{1}{2}\log|\boldsymbol{\Sigma}| - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})$$

X.MU → MEDIA

C → ARRAY [M,M] CHE RAPPRESENTA LA COVARIANZA

Write a function to compute the log-density `logpdf_GAU_ND(x, mu, C)` for a sample `x`. `mu` should be a numpy `array` of shape `(M, 1)`, whereas `C` is a numpy `array` of shape `(M, M)` representing the covariance matrix $\boldsymbol{\Sigma}$.

$C = \Sigma$

QUANDO POI VAI A CALCOLARE LA GAUSS ALLORA USI SOLO LA RIGA CORRISPONDENTE ALLA CLASSE

Suggestion 1: The inverse of a 2-D array `C` can be computed using `numpy.linalg.inv`. The log-determinant $\log|C|$ of `C` can be computed using `numpy.linalg.slogdet` — pay attention that the function returns **two** values, the absolute value of the log-determinant is the second one (the first is the sign of the determinant, which, for covariance matrices, is positive). Again, directly computing the logarithm of the determinant can cause numerical issues. `numpy.linalg.slogdet` performs the computations in a way that is robust also for very small values of $|C|$.

Suggestion 2: In some cases we will need to compute the log-density for a set of N samples $\boldsymbol{X} = [\boldsymbol{x}_1 \dots \boldsymbol{x}_N]$. It can be convenient to write the function `logpdf_GAU_ND` so that it takes as argument a $M \times N$ matrix `X` rather than a single sample `x`, and computes the vector of log-densities $Y = [\log\mathcal{N}(\boldsymbol{x}_1|\boldsymbol{\mu},\boldsymbol{\Sigma}) \dots \log\mathcal{N}(\boldsymbol{x}_N|\boldsymbol{\mu},\boldsymbol{\Sigma})]$. The term $\boldsymbol{x}_i$ here refers to the i-th sample, and corresponds to the i-th column of `X`. → È NECESSARIO PRENDERE PIÙ CAMPIONI (DELLO STESSO TIPO?)

Suggestion 3: Broadcasting and re-arranging of the computations can be used in this case to significantly speed up the computation of Y, and to avoid explicitly looping over the elements of $\boldsymbol{X}$. However, getting it right is non trivial. If you want to try you can ask for more insights. Otherwise, simply compute the array of log-densities using a loop over the columns of the data matrix `X`.

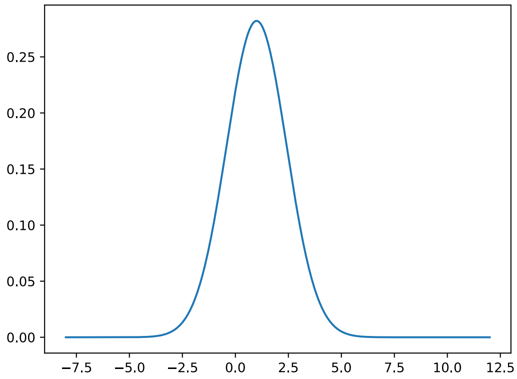
CALCOLARE LA MATRICE DELLE DENSITÁ LOGICHE UTILIZZANDO UN CICLO SULLE COLONNE DELLA MATRICE DI DATI X

In the following we assume that `logpdf_GAU_ND(X, mu, C)` takes as input a data matrix `X` and returns a 1-D numpy array of log-densities Y. You can visualize your density — in the example we plot the density for $\mu = 1, \Sigma = 2$ (remember that these should be 1×1 numpy arrays):

PER OGNI CLASSE

D

```
import matplotlib.pyplot as plt
plt.figure()
XPlot = numpy.linspace(-8, 12, 1000)
m = numpy.ones((1,1)) * 1.0
C = numpy.ones((1,1)) * 2.0
plt.plot(XPlot.ravel(), numpy.exp(logpdf_GAU_ND(vrow(XPlot), m, C)))
plt.show()
```



You can also check whether your density is correct by comparing your values with those contained in `Solution/llGAU.npy`

```
pdfSol = numpy.load('Solution/llGAU.npy')
pdfGau = logpdf_GAU_ND(vrow(XPlot), m, C)
print(numpy.abs(pdfSol - pdfGau).max())
```

The result should be zero or very close to zero (it may not be exactly zero due to numerical errors, however it should be a very small number, e.g. $\approx 10^{-17}$)

You can also check the density for the multi-dimensional case using the samples contained in `Solution/XND.npy`:

```
XND = numpy.load('Solution/XND.npy')
mu = numpy.load('Solution/muND.npy')
C = numpy.load('Solution/CND.npy')
pdfSol = numpy.load('Solution/llND.npy')
pdfGau = logpdf_GAU_ND(XND, mu, C)
print(numpy.abs(pdfSol - pdfGau).max())
```

Again, the result should be zero or close to zero.

Maximum Likelihood Estimate

The ML estimate for the parameters of a Multivariate Gaussian distribution correspond to the empirical dataset mean and the empirical dataset covariance

$$\boldsymbol{\mu}_{ML} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_i \ , \quad \boldsymbol{\Sigma}_{ML} = \frac{1}{N}\sum_{i=1}^{N} (\boldsymbol{x}_i - \boldsymbol{\mu})(\boldsymbol{x}_i - \boldsymbol{\mu})^T$$

These can be computed as in Laboratory 3. For the samples contained in `'Solution/XND.npy'` you should obtain

$$\boldsymbol{\mu}_{ML} = \begin{bmatrix} -0.07187197 \\ 0.05979594 \end{bmatrix} \ , \quad \boldsymbol{\Sigma}_{ML} = \begin{bmatrix} 0.94590166 & 0.09313534 \\ 0.09313534 & 0.8229693 \end{bmatrix}$$

MEDIA

We can also compute the log-likelihood for our estimates. The log-likelihood corresponds to the sum of the log-density computed over all the samples

$$\ell(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{i=1}^{n} \log \mathcal{N}(x_i|\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

SOMMA TUTTI QUELLI CHE HAI CALCOLATO PER OGNI SAMPLE

Write a function to compute the log-likelihood. Using the ML estimates (m_ML and C_ML in the following code) the log-likelihood should be

```
ll = loglikelihood(XND, m_ML, C_ML)
print(ll)
```

```
-270.70478023795044
```

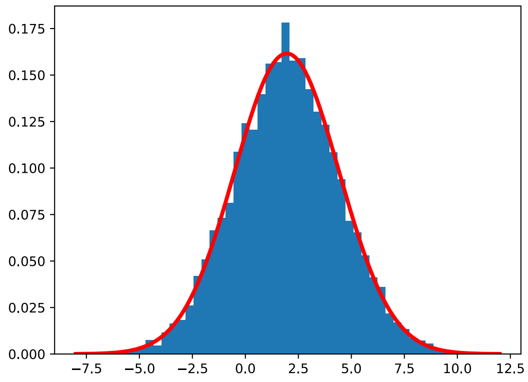
A second dataset corresponding to the file 'Solution/X1D.npy' contains one-dimensional samples. It can be loaded with numpy.load. For this dataset you should obtain

$$\boldsymbol{\mu}_{ML} = \begin{bmatrix} 1.9539157 \end{bmatrix} \quad , \quad \boldsymbol{\Sigma}_{ML} = \begin{bmatrix} 6.09542485 \end{bmatrix}$$

We can visualize how well the estimated density fits the samples plotting both the histogram of the samples and the density (again, m_ML and C_ML are the ML estimates):

```
plt.figure()
plt.hist(X1D.ravel(), bins=50, density=True)
XPlot = numpy.linspace(-8, 12, 1000)
plt.plot(XPlot.ravel(), numpy.exp(logpdf_GAU_ND(vrow(XPlot), m_ML, C_ML)))
```

where m and C are the ML estimates for the dataset



In this case, the log-likelihood for the ML estimates is

```
ll = loglikelihood(X1D, m_ML, C_ML)
print(ll)
```

```
-23227.077654602715
```

SE CAMBI QUESTI HAI VALORI MINORI

You can verify that computing the log-likelihood for other values of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ would results in a lower value of the log-likelihood.

Project

LO DEVI APPLICARE ALLE DIVERSE CARATTERISTICHE DELLE VARIE CLASSI, PER OGNI CLASSE, PER OGNI COMPONENTE DEL VETTORE DELLE CARATTERISTICHE DI QUELLA CLASSE' CALCOLA ML

DENSITY = TRUE

Try fitting uni-variate Gaussian models to the different features of the different classes of the project dataset. For each class, for each component of the feature vector of that class, compute the ML estimate

SE IL VALORE DI LOG-LIKELIHOOD È ALTO VUOL DIRE CHE C'È UN BUON ADATTAMENTO

for the parameters of a 1D Gaussian distribution. Plot the distribution density (remember that you have to exponentiate the log-density) on top of the normalized histogram (set `density=True` when creating the histogram, see Laboratory 2). What do you observe? Are there features for which the Gaussian densities provide a good fit? Are there features for which the Gaussian model seems significantly less accurate?

Note: for this part of the project, since we are still performing some preliminary, qualitative analysis, you can compute the ML estimates and the plots either on the whole training set. In the following labs we will employ the densities for classification, and we will need to perform model selection, therefore we will re-compute ML estimates on the model training portion of the dataset only (see Laboratory 3).

IN QUESTO CASO DEVONO ESSERE CALCOLATE SU TUTTI I DATI

. SONO 2 CLASSI
. 6 FEATURES

TUTTE LE INFO PER

0 1

F1
F2
F3
F4
F5
F6

DEVI PRENDERE LA RIGA DI TUTTI I VERI E FALSO

PER OGNI FEATURE
HIST PER (0,1) INSIEME

Sì, la curva disegnata sul grafico rappresenta la densità di probabilità gaussiana calcolata a partire dai dati. Se i dati seguono effettivamente una distribuzione gaussiana, allora la curva dovrebbe adattarsi bene all'istogramma dei dati. In altre parole, la forma dell'istogramma dovrebbe essere simile a quella della curva gaussiana. Se ciò accade, possiamo dire che il modello gaussiano è un buon adattamento per i dati. Se invece la curva gaussiana non si adatta bene all'istogramma, allora il modello gaussiano potrebbe non essere il modello più appropriato per quei dati.

I valori di log-verosimiglianza più alti indicano un adattamento migliore del modello ai dati. Quindi, per ogni caratteristica, il modello si adatta meglio ai dati del gruppo (True o False) che ha il valore di log-verosimiglianza più alto. Dai risultati che hai fornito, possiamo vedere che:
Per la caratteristica 1, il modello si adatta meglio ai dati del gruppo False.
Per la caratteristica 2, il modello si adatta meglio ai dati del gruppo True.
Per la caratteristica 3, il modello si adatta meglio ai dati del gruppo False.
Per la caratteristica 4, il modello si adatta meglio ai dati del gruppo False.
Per la caratteristica 5, il modello si adatta meglio ai dati del gruppo False.
Per la caratteristica 6, il modello si adatta meglio ai dati del gruppo False.
Quindi, sembra che per la maggior parte delle caratteristiche, il modello gaussiano si adatti meglio ai dati del gruppo False.

Il confronto va fatto all'interno di ogni feature distinto fra le diverse classi nella stessa feature