From DTU Learn download the scripts for this week via the link called 02450Toolbox_Python_all. It contains all the scripts for the course (incl. the ones we have already covered, so be careful not to overwrite modified scripts etc. This is the last time you need to download scripts....

Probability and probability densities with PYTHON

Objective: Upon completing this exercise it is expected that you:

- Understand the univariate and multivariate normal distribution.

Material: Lecture notes *"Introduction to Machine Learning and Data Mining"* as well as the files provided from DTU Learn.

4.1 Understanding the Univariate and Multivariate Normal Distribution

The univariate and multivariate normal distribution is central to many machine-learning methods. In this exercise we will investigate the basic properties of these distributions in Python.

4.1.1 Inspect and run the script `ex4_1_1.py`. The script generates 200 samples from a Normal distribution with mean $\mu = 17$ and standard deviation $\sigma = 2$ and plot the samples as well as a histogram of the samples.

Script details:

- *Read about* `np.random.normal()` *function to learn how to generate Normal distributed random numbers in Python.*
- *The function* `plot()` *can be used to plot the samples.*
- *The function* `hist()` *can be used to plot a histogram.*
- *Check Matplolib tutorial for more examples:* *matplotlib.sourceforge.net/users/pyplot_tutorial.html*

Try running the code a few times and see how the data and histogram changes when new random numbers are generated from the same distribution.

The histogram is generated by counting how many of the samples fall within the range covered by each bin of the histogram. Try changing the number of bins in the histogram.

4.1.2 Compute the empirical mean and standard deviation of the generated samples. Show, that they are close but not equal to the theoretical values used to generate the random samples. See the script `ex4_1_2.py` for details.

Script details:

- *Take a look at the functions* `mean` *and* `std`.

Try running the code a few times and see how the empirical mean and standard deviation changes when new random numbers are generated from the same distribution.

In the next part we will see how the number of samples influence the resulting histogram.

4.1.3 Inspect and run the script `ex4_1_3.py` which generates random samples from the Normal distribution and plots the histogram as before. Also, the function plots the true probability density function (`scipy.stats.norm.pdf()`) of the Normal distribution.

Show that when the number of samples `N` is increased, the histogram approximates the pdf better and the empirical estimates of the mean and standard deviation improve.

So far we have been considered a 1-dimensional Normal distributed random variable. In the next step we will consider the multivariate Normal distribution in two dimensions.

The covariance matrix for a 2D Gaussian is described by

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_1^2 & \text{cov}\,(x_1, x_2) \\ \text{cov}\,(x_2, x_1) & \sigma_2^2 \end{bmatrix},$$

where $\text{cov}(\cdot, \cdot)$ is covariance between two random variables. Note that $\text{cov}\,(x_1, x_2) = \text{cov}\,(x_2, x_1)$, i.e., the covariance matrix is symmetric, and $\sigma_n^2 = \text{cov}\,(x_n, x_n)$. We can write the covariance matrix in terms of Correlation, i.e.

$$\begin{aligned} \text{Correlation}\,(x_1, x_2) &= \frac{\text{cov}\,(x_1, x_2)}{\sqrt{\text{cov}\,(x_1, x_1)}\sqrt{\text{cov}\,(x_2, x_2)}} \Rightarrow \\ \text{cov}\,(x_1, x_2) &= \text{Correlation}\,(x_1, x_2)\sqrt{\text{cov}\,(x_1, x_1)}\sqrt{\text{cov}\,(x_2, x_2)}. \end{aligned}$$

4.1.4 Inspect and run the script `ex4_1_4.py`. The script generates 1000 samples from a 2-dimensional Normal distribution with mean

$$\boldsymbol{\mu} = \begin{bmatrix} 13 & 17 \end{bmatrix}$$

and covariance matrix

$$\mathbf{\Sigma} = \begin{bmatrix} 4 & 3 \\ 3 & 9 \end{bmatrix}.$$

Script details:

- *Look at the function* `np.random.multivariate_normal()` *to learn how you can generate multivariate Normal distributed random numbers in Python.*

4.1.5 Inspect and run the script `ex4_1_5.py` which generates 2-dimensional Normal distributed random samples and plots a scatter plot as well as a 2-dimensional histogram. In the script, the covariance matrix is constructed from the standard deviations and correlation as described above.

Show that when the correlation between x_1 and x_2 is zero, the scatter plot and 2-d histogram have the shape of an axis-aligned ellipse. Can you explain why?

Show that when the correlation between x_1 and x_2 is one, the values of x_1 and x_2 fall on a straight line. Can you explain why?

Try varying the number of samples, the mean, the standard deviations, the correlation and the number of histogram bins and see what happens.

In this part we will use the concepts and commands from the previous section in order to make some very simple statistical models of the digits data set (16×16 pixel images of hand written digits) we considered in last weeks exercise.

4.1.6 Inspect and run the script `ex4_1_6.py`. The script loads the digits data set, and computes the mean of each pixel, the standard deviation of each pixel, and the covariance matrix between the pixels. By default, only images of "ones" are included in the analysis.

Does the mean look like you expect? Can you explain why the standard deviation is highest along the edges of the digit one? Try to change the digit you analyze and get a felling of how different the individual digits are.

For each pixel we now have a mean and a standard deviation, i.e., 256 means and 256 corresponding standard deviations. So in essence we can make a a simple model of the digits with a Normal distributions for each individual pixel.

Since we know how to draw a new sample from a Normal distribution we can draw a sample for each individual pixel based on their respective 1D normal distribution (i.e. draw a total of 256 values). Combining these samples we end up with a new digit. The question is now how natural our newly generated/artificial samples are, and if they at all are possible to recognize as digits.

With our simple model above we argued that we had 256 different 1D Normal distributions; however, we could also look at the problems in terms of the multivariate Normal. Instead of assuming 256 independent 1D Gaussians we could formulate our model for digits as a 256-dimensional multivariate Normal, which allows each pixel to depend on the other pixels. This dependency is described through the covariance matrix.

4.1.7 Inspect and run the script `ex4_1_7.py`. The script generates 10 new images with the same mean and standard deviation as the data using a 1-dimensional Normal distribution for each pixel. Next, the script generates 10 new images with the same mean and covariance as the data using a $16 \cdot 16 = 256$-dimensional multivariate Normal distribution.

Which model is best? Try changing the analyzed digits and see what happens.

4.1.8 (Extra challenge) Try to vary the number of observations of a given digit you use to estimate the mean and (co)variance. Does the number of observations used make a difference on the quality of the generated digits?

4.2 Tasks for the report

If you havn't already, you can now write the final part of the report.

- **A discussion explaining what you have learned about the data.**
  Summarize the most important things you have learned about the data and give your thoughts on whether your primary machine learning aim appears to be feasible based on your visualization.

References