Probability densities and data Visualization with PYTHON

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

- Understand the univariate and multivariate normal distribution.
- Get an understanding of the many ways data can be visualized including histograms, boxplots, and scatter plots.

Material: Lecture notes *"Introduction to Machine Learning and Data Mining"* as well as the files in the exercise 4 folder available from Campusnet.

Piazza discussion forum: You can get help by asking questions on Piazza: https://piazza.com/dtu.dk/fall2020/02450

Software installation: Extract the Python toolbox from DTU Inside. Start Spyder and add the toolbox directory (`<base-dir>/02450Toolbox_Python/Tools/`) to `PYTHONPATH` (Tools/PYTHONPATH manager in Spyder). Remember the purpose of the exercises is not to re-write the code from scratch but to work with the scripts provided in the directory `<base-dir>/02450Toolbox_Python/Scripts/` Representation of data in Python:

	Python var.	Type	Size	Description
	`X`	numpy.array	$N \times M$	Data matrix: The rows correspond to N data objects, each of which contains M attributes.
	`attributeNames`	list	$M \times 1$	Attribute names: Name (string) for each of the M attributes.
	`N`	integer	Scalar	Number of data objects.
	`M`	integer	Scalar	Number of attributes.
Classification	`y`	numpy.array	$N \times 1$	Class index: For each data object, `y` contains a class index, $\texttt{y}_n \in \{0, 1, \ldots, C-1\}$, where C is the total number of classes.
Classification	`classNames`	list	$C \times 1$	Class names: Name (string) for each of the C classes.
Classification	`C`	integer	Scalar	Number of classes.

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

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

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

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

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 Visualizing Fisher's Iris data

In this exercise we will reproduce most of the figures in "Introduction to Data Mining." section 3.3 in Matlab using the Iris flower dataset that was also introduced in Exercise 1.

4.2.1 The Iris data set is available in the Excel file `Data/iris.xls`. Load the data into Python and generate all the variables described in *Representation of data in Python* using the script `ex4_2_1.py`.

Script details:

- *You can use the package* `xlrd` *which you have used before in exercise 2.2.*

4.2.2 Inspect and run `ex4_2_2.py`. The script plots a histogram of each of the four attributes in the Iris data.

Script details:

- *You can use the command* `hist` *to plot a histogram.*
- *Use indexing to extract each attribute. For example,* `X[:,m-1]` *extracts the* `m`*'th attribute.*
- *For multiple plots in one figure window you can use the command* `subplot(n,m,i)`.

Show on the graph that the petal length is either between 1 and 2 cm. or between 3 and 7 cm. but that no flowers in the data set have a petal length between 2 and 3 cm. Do you think this could be useful to discriminate between the different types of flowers?

4.2.3 Inspect and run `ex4_2_3.py`. The script produces a boxplot of the four attributes in the Iris data as shown in Figure 3.11 in the book.

Script details:

- *Take a look at the function* `boxplot`.
- *Type* `help(boxplot)`*" to see how you can adjust the boxplot and add labels.*

This boxplot shows the same information as the histogram in the previous exercise. Discuss the advantages and disadvantages of the two types of plots.

4.2.4 Inspect and run `ex4_2_4.py`. The scripts produces a boxplot for each attribute for each class as shown in Figure 3.12 in the book.

Script details:

- *Use the functions* `subplot()` *and* `boxplot()`.
- *The variable* `y` $\in \{0, 1, \ldots, C-1\}$ *contains the class labels. To extract the data objects belonging to, say, class* `c`*, you can use* `y` *to index into* `X` *like this:* `X[(y==c), :]`
- *It is easier to compare the boxplots if they are all on the same axis. To do this, you can use the function* `ylim()`.

Show on the graph that all the Iris-setosa in this data set have a petal length between 1 and 2 cm.

4.2.5 Inspect and run `ex4_2_5.py`. The scripts produces a matrix of scatter plots of each combination of two attributes against each other as shown in Figure 3.16 in the book.

Script details:

- *To make a scatter plot, you can use the function* `plot(x,y,s)` *where* `x` *and* `y` *specify the coordinates and* `s` *is a string that specifies the line style and plot symbol, e.g.,* `s='.'` *to make dots.*
- *To extract the data values for the* `m`*'th attribute in the* `c`*'th class, you can write* `X[(y==c),m]`.
- *You can use the command* `hold all` *to plot multiple plots on top of each other.*

Say you want to discriminate between the three types of flowers using only the length and width of either sepal or petal. Show on the graph why it would be better to use petal length and width rather than sepal length and width.

4.2.6 Inspect and run `ex4_2_6.py`. The script produces a 3-dimensional scatter plot of three attributes as shown in Figure 3.17 in the book.

Script details:

- *Read more about plotting in 3 dimensions:* `matplotlib.sourceforge.net/mpl_toolkits/mplot3d/tutorial.html`
- *To plot in 3 dimensions you need to import* `matplotlib.pyplot` *as earlier, and additionally* `Axes3D` *from* `mpl_toolkits.mplot3d`.

Try rotating the data. Can you find an angle where the three types of flower are separated in the plot?

4.2.7 Use the script `ex4_2_7.py` to plot the data matrix as an image as shown in Figure 3.23 in the book. The data matrix should be standardized to have zero mean and unit standard deviation.

Script details:

- *You can use the function* `imagesc()` *to plot an image.*
- *By default, the image will be smoothed, what is not always desired when you look at the data. Use parameter* `interpolation='None'` *to display raw data.*
- *The function* `zscore()` *can be used to standardize the data matrix.*

You are welcome to try out other plotting methods for the data. Matplotlib online repository is a good source of inspiration:
`matplotlib.sourceforge.net/gallery.html`

4.3 Visualizing Wine Data

We will in this part of the exercise consider two datasets related to red and white variants of the Portuguese "Vinho Verde" wine [1], the data has been downloaded from `http://archive.ics.uci.edu/ml/datasets/Wine+Quality`. Only physico-chemical and sensory attributes are available, i.e., there is no data about grape types, wine brand, wine selling price, etc. The data has the following attributes:

Attributes 1–11 are based on physicochemical tests and attribute 12 on human judging. Later in the course we will attempt to predict whether a wine is a red or

#	Attribute	Unit
1	Fixed acidity (tartaric)	g/dm^3
2	Volatile acidity (acetic)	g/dm^3
3	Citric acid	g/dm^3
4	Residual sugar	g/dm^3
5	Chlorides	g/dm^3
6	Free sulfur dioxide	mg/dm^3
7	Total sulfur dioxide	mg/dm^3
8	Density	g/cm^3
9	pH	pH
10	Sulphates	g/dm^3
11	Alcohol	% vol.
12	Quality score	0–10

white wine based on these attributes and we will also attempt to predict the wine quality. Unfortunately, the data set has many observations that can be considered outliers and in order to carry out analyses later it is important to remove the corrupt observations.

The aim of this exercise is to use visualization to identify outliers and remove these outliers from the data. It might be necessary to remove some outliers before other outlying observations become visible. Thus, the process of finding and removing outliers is often iterative. The wine data is stored in a Matlab file, `Data/wine.mat`.

4.3.1 Inspect and run the script `ex4_3_1.py`. The scripts loads the data into Python using the `scipy.io.loadmat()` function, as in previous exercises. This dataset contains many observations that can be considered outliers and the visualization tools you have worked with in the previous exercise is used to identify the outliers in the data set. How many outliers are identified by the script? How are the identified outliers removed from the data set?

Script details:

- *You can use your solutions to the previous exercise as a starting point for making your visualizations.*
- *Say you want to find all data objects for which the alcohol percentage (attribute number 11) is not greater than 100%. You can mask them simply as* `mask=(X[:,10]<=100)`.
- *You can use the mask to eliminate the outlier observations (rows of data matrix). For instance you can write* `X=X[mask,:]` *where* `mask` *indicates the data objects that should be maintained. Remember also to remove them from the class index vector,* `y=y[mask,1]` *and to recompute* `N`.

We will later in the course attempt to classify the type of wine (white or red) as well as predict the quality of wine based on the physicochemical tests. Visual inspection of the data can give an indication of the difficulty of these tasks.

4.3.2 Are there any of the measurements that seem to be well suited in order to discriminate between red and white wines? What plots are particular useful in order to investigate this?

4.3.3 Does any of the 12 attributes appear to correlate with each other? What plots are well suited to investigate this?

4.3.4 Can you identify any clear relationship between the various physicochemical measurements of the wines and the quality of the wines as rated by human judges?

4.4 Tasks for the report

After today, you can address the last questions for part one of the report. Note if you have categoric variables you can still analyze these by PCA and other modeling approaches that assumes interval or ratio data types by converting them to one-out-of-K coding. The function `categoric2numeric` converts a column vector of a categoric attribute to numeric using one-out-of-K coding, type help(categoric2numeric) to learn more.

- **Data visualization(s) based on suitable visualization techniques including a principal component analysis (PCA).**
  Touch upon the following subjects, use visualizations when it appears sensible. *Keep in mind the ACCENT principles and Tufte's guidelines when you visualize the data.*
  - Are there issues with outliers in the data,
  - do the attributes appear to be normal distributed,
  - are variables correlated,
  - does the primary machine learning modeling aim appear to be feasible based on your visualizations.
- **A discussion explaining what you have learned about the data.**
  Summarize here the most important things you have learned about the data and give also your thoughts on whether your primary machine learning aim appears to be feasible based on your visualization.

References

[1] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. Modeling wine preferences by data mining from physicochemical properties. *In Decision Support Systems, Elsevier*, 47(4):547–553, 2009.