Principal Component Analysis with PYTHON

Objective: To get acquainted with how data can be filtered and visualized using principal component analysis (PCA). Upon completing this exercise it is expected that you:

- Can apply and interpret principal component analysis (PCA) for data visualization.

Material: Lecture notes *"Introduction to Machine Learning and Data Mining"* as well as the files in the exercise 2 folder available from Campusnet.

PYTHON Help: You can get help in your Python interpreter by typing `help(obj)` or you can explore source code by typing `source(obj)`, where `obj` is replaced with the name of function, class or object.
Furthermore, you get context help in Spyder after typing function name or namespace of interest. In practice, the fastest and easiest way to get help in Python is often to simply Google your problem. For instance: `"How to add legends to a plot in Python"` or the content of an error message. In the later case, it is often helpful to find the *simplest* script or input to script which will raise the error.

Piazza discussion forum: You can get help by asking questions on Piazza:
https://piazza.com/dtu.dk/fall2019/02450

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/` For today's exercises you need to a package to your Python environment. The additional package is a machine learning toolkit for the last exercise (today optionally, but we shall need it in the following weeks). Please make sure that you have installed the following package (you can follow the guidelines at the corresponding websites):

- Machine learning toolkit (scikit-learn) - large package implementing various ML methods for supervised and unsupervised learning:
  `http://scikit-learn.org/stable/install.html`

The websites provide documentation of the packages. Note if you use the Anaconda Python distribution these packages may already be added, use `conda list` in the terminal for a list of installed packages.

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, $\mathtt{y}_n \in \{0, 1, \dots, C-1\}$, 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.

2.1 PCA on the Nanose dataset

As an example dataset we will consider chemical sensor data obtained from the NanoNose [1] project, see also [2]. The data contains 8 sensors named by the letters *A*–*H* measuring different levels of concentration of Water, Ethanol, Acetone, Heptane and Pentanol injected into a small gas chamber. The data will be represented in matrix form such that each row contains the 8 sensors measurements (i.e. sensor A-H) of the various compounds injected into the gas chamber.

2.1.1 Inspect the file `<base-dir>/02450Toolbox_Python/Data/nanonose.xls` and make sure you understand how the data is stored in Excel. We will load the Nanose dataset from the file `<base-dir>/02450Toolbox_Python/Data/nanonose.xls` into into Python using the `xlrd` package, and get it into the standard data matrix form as we learnt how to do it in Exercise 1. See `ex2_1_1.py` for details. There are 90 data objects with 8 attributes each. Do you get the correct data matrix X of size 90×8?

2.1.2 The data resides in an 8 dimensional space where each dimension corresponds to each of the 8 NanoNose sensors. This makes visualization of the raw data difficult, because it is difficult to plot data in more than 2–3 dimensions.

Plot the two attributes A and B against each other in a scatter plot using `ex2_1_2.py`.

Script details:

- *You need to import* `matplotlib.pyplot` *package to use plotting functions in Python:*
  ```
  from matplotlib.pyplot import *
  ```

- *Use* `plot()` *function to plot data.*
- *The attributes A and B are the first and second columns of the matrix $\boldsymbol{X}$.'*
- *You can use indexing to get the columns out of the matrix, e.g.,* `x=X[:,1]` *or* `y = X[:,2]`
- *Notice that the third argument of the* `plot()` *command can be used to set a plot symbol. For example, the command* `plot(x,y,'o')` *plots a scatter plot with circles.*
- *Use* `show()` *function to render the plot.*
- *You can find extensive help and numerous examples on matplotlib website:* `http://matplotlib.sourceforge.net`

Try to change the dimensions that are plotted against each other.

We will use principal component analysis to reduce the dimensionality of the data. PCA is computed by subtracting the mean of the data, $\boldsymbol{Y} = \boldsymbol{X} - \mathbf{1}\boldsymbol{\mu}$ (where $\boldsymbol{\mu}$ is a (row) vector containing the mean value of each attribute and $\mathbf{1}$ is a N by 1 column vector of ones in all entries) and then calculating the singular value decomposition (SVD) of the zero mean data, i.e. $\boldsymbol{Y} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top$.

From PCA we can find out how much of the variation in the data each PCA component accounts for. This is given by

$$\rho_m = \frac{s_{mm}^2}{\sum_{m'=1}^{M} s_{m',m'}^2},$$

i.e. the squared singular value of the given component divided by the sum of all the squared singular values.

2.1.3 Compute the PCA of the NanoNose data and plot the percent of variance explained by the principal components as well as the cumulative variance explained using `ex2_1_3.py`.

Script details:

- *You can use the method* `mean()` *of array or matrix object to compute the mean of the data. You should compute the mean for each attribute(column), i.e., the vector of means should have M elements.*
- *You cannot directly subtract a vector from a matrix. One way to accomplish this is to subtract the product of vector of ones and vector of means:* `Y = X - np.ones((N,1))*X.mean(0)`
- *You can use the function* `numpy.linalg.svd()` *to compute the SVD.*
- *To extract the diagonal from a matrix, use the method* `diagonal()` *of an array object, or use* `np.diagonal()` *or* `np.diag()`.

Can you verify that more than 90% of the variation in the data is explained by the first 3 principal components? How many components would be needed for 95 %?

2.1.4 Plot principal component 1 and 2 against each other in a scatterplot, see the script `ex2_1_4.py` for details.

Script details:

- *Data can be projected onto the principal components using* `Z = Y@V` *or* `Z = np.dot(Y,V)`, *where* `Y` *is centered data.*
- *You learned how to make a scatter plot in Exercise 2.1.2.*

What are the benefits of visualizing the data by the projection given by PCA over plotting two of the original data dimensions against each other? Compare with the scatter plots of attributes you made in Exercise 2.1.2.

2.1.5 Interpret the principal directions ($\boldsymbol{V}$) obtained using the PCA. Consider the script `ex2_1_5.py`. Which of the original attributes does the second principal component mainly capture the variation of and what would cause an observation to have a large negative/positive projection onto the second principal component? (remember both the attributes and the prinpal component has a sign and a magnitude)

Script details:

- *The columns of* `V` *gives you the principal component directions*
- *The data is projected onto the second principal component by* `Y@V[:,1]`

We can correct for differences in scale by standardization. When doing PCA on data with attributes of different scales, it can be very important to standardize the dataset. We standardize a dataset by ensuring each attribute has a mean of zero (as before), but also has a variance of one (i.e. zero mean and unit variance).

In `ex2_1_5.py` you saw that we can interpret the principal directions by investigating the coefficients in the vectors of $\boldsymbol{V}$. Another way to approach intepreting the principal directions is to plot the coefficients as vectors in the principal component space. In the PC1/PC2-space, we can for instance interpret the relationship between PC1, PC2 and a given attribute by drawing a line form Origo to the coefficients in PC1 and PC2 corresponding to the attribute. The diretion and magnitude of such a vector defines how the data from that attribute is projected onto the PC1/PC2-space—e.g. if the vector points in positive direction of PC1, then positive values of that attribute contributes to a positive projection onto PC1. Since the vectors in $\boldsymbol{V}$ are unit-vectors, all coefficients will lie within the unit-circle.

2.1.6 Investigate the standard deviation of the NanoNose attributes and try to determine if some of the attributes have higher variance than the others using `ex2_1_6.py`. Which attribute has the highest standard deviation? Use the script to visualize the difference between either only subtracting the mean or both subtracting the mean and dividing by the standard deviation (visualize: the projection, attribute coefficients, and the variance explained of a PCA for the two). How did the attribute with the highest standard deviation change in terms of its direction and magnitude in the attribute coefficients? How did the variance explained change? Lastly, try multiplying one of the attributes with a factor 100 and see how that changes the PCA.

2.2 Structure in handwritten digits

The US Postal Service (USPS) wanted to automate the process of sorting letters based on their zip-codes. We will presently consider a dataset of USPS handwritten digits available at `http://www.cad.zju.edu.cn/home/dengcai/Data/MLData.html`, see also [3]. There are two datasets containing handwritten digits `testdata` and `traindata`.

2.2.1 Load the dataset. Inspect and run the script `ex2_2_1.py` to visualize the first digit of the traindata (the script uses `reshape` to turn a digit vector into an image and `imshow()` to display the image).

2.2.2 Inspect and run the script `ex2_2_2.py`. Show that it requires 22 PCA components to account for more than 90% of the variance in the data. Show that the first principal component is almost sufficient to separate zeros and ones. Examine the first principal component and discuss and understand what it captures.

2.2.3 Change the value of `K` and show that reconstruction accuracy improves when more principal components are used. How many principal components do you need to be able to see the different digits properly? What happens if you set `K=256`?

2.2.4 Try decomposing one digit at a time. Hint: Modify the variable `n` to contain only a single digit. Explain what happens to the principal components when only a single digit type is analyzed compared to when all digit types are analyzed at the same time.

2.3 Extra challenge

We will later in the course learn various methods for classification. Among the approaches we will learn is K-nearest neighbor (KNN) classification. For now we will consider the KNN classifier a black box that we will use to evaluate how well we can determine the digit class in the space given by the K first principal components, i.e. after filtering out the PCA components with smallest singular values which we consider components pertaining to noise.

2.3.1 Inspect and run the script `ex2_3_1.py` and see how well we are able to classify the digits when we use say `K=10` PCA components, `K=40` PCA components and the whole data, i.e `K=256` PCA components. Show that the classifier is best when using around 40–60 PCA components, and explain why that is so.

2.4 Tasks for the report

After today's exercise you should be able to explain what types of attributes are in your data (i.e. discrete/continuous, Nnminal/ordinal/interval/ratio, see also today's lecture) as well as be able to apply and interpret the results of a principal component analysis (PCA) of your data. Notice, that there are three aspects that needs to be described in the PCA analysis for the report:

- The amount of variation explained as a function of the number of PCA components included,
- the principal directions of the considered PCA components,
- the data projected onto the considered principal components.

References

[1] Nanonose project.

[2] Tommy S Alstrøm, Jan Larsen, Claus H Nielsen, and Niels B Larsen. Data-driven modeling of nano-nose gas sensor arrays. In *SPIE Defense, Security, and Sensing*, pages 76970U–76970U. International Society for Optics and Photonics, 2010.

[3] Jonathan J. Hull. A database for handwritten text recognition research. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 16(5):550–554, 1994.