Machine Learning
Winter Semester 2019/2020

Sheet #1 – Solution

Due: 24.10.2019, 16:00

Fred Hamprecht
Manuel Haußmann

1 Principal Component Analysis (10pt)

In the lecture we derived how to use the PCA approach to find a "best" linear one-dimensional subspace to summarize high dimensional data for visualization and compression purposes. In this exercise, you will implement this approach in python to get more comfortable with the concept. Assume we have a data set consisting of N observations with p features, summarized in a (centered) data matrix $\mathbf{X} \in \mathbb{R}^{p \times N}$. As was mentioned in the lecture and as you can show in the bonus part **(c)**, the principal components correspond to the eigenvectors of the matrix $\mathbf{X}\mathbf{X}^T$ ordered by the size of their eigenvalues.

(a) PCA for visualization (6pt). One major use case of PCA is for data visualization in order to get an idea of the structure of some high dimensional data. Implement PCA in python using `numpy`. Use your implementation to then map the protein data we discussed in the lecture `protein.txt`[1] to 2d, plot and discuss any patterns you observe.
Hint: You may use ***`numpy.linalg.eig`*** *in order to compute eigenvalues/-vectors.*
Hint: In order to know which point corresponds to which country have a look at how to annotate plots in matplotlib[2].

Solution: See the jupyter notebook.

(b) PCA for lossy compression (4pt). A second major use case of PCA is for data compression. It can be shown that the distortion, i.e. the error introduced by these mappings, is the sum of the "dropped" eigenvalues. Mapping into a M dimensional subspace gives a distortion of

$$\text{error}_M = \sum_{m=M+1}^{p} \lambda_m.$$

i) Use your PCA implementation to compute the principal components of the `protein` data set and plot the eigenvalues ordered by size (i.e. the scree plot), as well as the error as a function of M.

ii) (Bonus) To visualize the loss of information, we will rely on images of the digit three, a subset of the famous MNIST digits[3], which you can access as `digits.npy`. Visualize the loss of information that result by first mapping the data to M principal components and back to the original space. Explore and discuss the changes as you move to higher and higher values of M (i.e. move right on the eigenvalue plot).
Hint: How to do that? An gray-scale image $\mathbf{I}$ *consisting of* $h \times w$ *pixels* ($\mathbf{I} \in \mathbb{R}^{h \times w}$) *can also be considered as a* $h \cdot w$ *dimensional vector. Use* ***`numpy.reshape`*** *to transform it for the PCA analysis and then reshape it back into a matrix for plotting.*

Solution: See the jupyter notebook.

(c) (Bonus) Extension to multiple principal components. In the lecture it was shown that the first principal component $\mathbf{w}_1$ is given by the eigenvector with the largest eigenvalue λ_1 of $\mathbf{X}\mathbf{X}^T$. Show via induction that the first M principal components are given by $\mathbf{w}_1, ..., \mathbf{w}_M$, given by the M largest eigenvalues.
Hint: Assume that it holds for the first $M - 1$ *eigenvectors. Then use Lagrange multipliers to ensure* $\mathbf{w}_M$ *is orthogonal to* $\mathbf{w}_m$ $(m < M)$ *and* $||\mathbf{w}_M||_2 = 1$*, and follow the argumentation from the lecture.*

[1] Source: http://www.dm.unibo.it/~simoncin/Protein.html. Load via `numpy.loadtxt('protein.txt')`
[2] https://matplotlib.org/3.1.1/gallery/text_labels_and_annotations/annotation_demo.html
[3] Source: http://yann.lecun.com/exdb/mnist/

Solution: Let $\mathbf{X} \in \mathbb{R}^{p \times N}$ be the centered data matrix. We know from the lecture that the first pc $\mathbf{w}_1$ is given via

$$\mathbf{X}\mathbf{X}^T\mathbf{w}_1 = \lambda_1 \mathbf{w}_1 \qquad \Leftrightarrow \qquad \mathbf{w}_1^T\mathbf{X}\mathbf{X}^T\mathbf{w}_1 = \lambda_1, \tag{1}$$

i.e. the covariance matrix $\mathbf{X}\mathbf{X}^T$ (spread) is maximized if we take λ_1 as the maximal eigenvalue.
Now assume we have it for $M-1$. Constraints on $\mathbf{w}_M$ are $||\mathbf{w}_M||_2 = 1$ and $\mathbf{w}_M^T\mathbf{w}_m = 0$ for all $m < M$. This gives us, using $\lambda_M, \mu_1, ..., \mu_{M-1}$ as the Lagrange multipliers the function to be maximized

$$\mathbf{w}_M^T\mathbf{X}\mathbf{X}^T\mathbf{w}_M + \lambda_M(1 - \mathbf{w}_M^T\mathbf{w}_M) + \sum_{m=1}^{M-1} \mu_m \mathbf{w}_M^T\mathbf{w}_m. \tag{2}$$

Setting the derivative wrt $\mathbf{w}_M$ to zero, gives

$$0 = 2\mathbf{X}\mathbf{X}^T\mathbf{w}_M - 2\lambda_M\mathbf{w}_M + \sum_{m=1}^{M-1} \mu_m \mathbf{w}_m. \tag{3}$$

Leftmultiplying with $\mathbf{w}_M^T$ and using the orthogonality constraint gives

$$\mathbf{w}_M^T\mathbf{X}\mathbf{X}^T\mathbf{w}_M = \lambda_M. \tag{4}$$

To maximize it we take the largest still remaining eigenvalue/-vector pair, getting the desired solution using our argumentation from above.

2 Kernel Density Estimation (10pt)

In the lecture, you learned about KDE. In this exercise, you will explore the influence of different kernels in exploring an unknown density you only have access to through samples.

(a) Kernel implementation (3pt). Implement a Quartic (biweight) kernel

$$k\left(x - \mu, w\right) = \frac{15}{16w}\left(1 - \left(\frac{x-\mu}{w}\right)^2\right)^2 \qquad \text{with support in } [-1, 1]$$

and plot it for $\mu = 0$ and $w = 1$ over the range $[-1, 1]$.
Hint: Use the fact that numpy works optimally with vectorized code, i.e. you can compute it over the whole range without a loop (see e.g. `?numpy.arange` *on how to get a vector over the whole range).*

Solution: See the jupyter notebook.

(b) Kernel density estimation (7pt). Take the first $N = 50$ data points from `samples.npy`, compute and plot the kernel density estimate over the range $[-10, 20]$ for a set of different bandwidths (e.g. $w = [0.1, 0.5, 1, 3, 5]$). Remember that the kernel density estimate is given as

$$\hat{p}(x) = \frac{1}{N}\sum_{n=1}^{N} k(x - x_n, w) = \frac{1}{Nw}\sum_{n=1}^{N} K\left(\frac{x - x_n}{w}\right).$$

Discuss the results and the influence of the bandwidth. Which bandwidth is optimal in your opinion? Explore what happens as you increase the number of samples N.[4]

[4]Some authors prefer the notation in the second equality, defining the biweight kernel as e.g. $K(u) = \frac{15}{16}(1-u^2)^2\mathbb{1}(|u| < 1)$ and pulling the $1/w$ normalization out of the kernel and in front of the sum.

Solution: See the jupyter notebook.

(c) (Bonus) Influence of the samples. In the lecture, we discussed that you have access to multiple sets of samples from the true distribution, i.e. assume that you have S set of N data points from the true distribution. Plot your estimates for each of the set in a single figure and discuss the results. In which regions do your samples agree, and where do they disagree. Are there patterns?
Hint: Use `plt.plot(xsample, estimate, color='darkblue', alpha=0.2)` *in order to plot each of the estimates. The alpha value makes the color of each line lighter and ensures that the average plot is darker the more individual estimates agree.*

Solution: See the jupyter notebook.