# Homework 3: SVD Imputation & Kernel PCA

Dr. Michael Moor
michael.moor@bsse.ethz.ch

Leslie O'Bray
leslie.obray@bsse.ethz.ch

Christian Bock
christian.bock@bsse.ethz.ch

Prof. Dr. Karsten Borgwardt
karsten.borgwardt@bsse.ethz.ch

*Submission deadline: 21.04.2021 at 14:00*

## Objectives

The goals of this homework are:

- to compute the optimal rank-r approximation for data imputation.
- to implement Kernel PCA using SVD in `Python`.

## Exercises

### Exercise 1: Data Imputation via SVD

In the last tutorial (see *Tutorial 3*) we looked at different applications of SVD, including data imputation using SVD. As we have seen in the tutorial we can impute missing data points in a data matrix $D$ using SVD and a rank-$r$ approximation $D_r$ of $D$. The SVD of a data matrix $D$ is defined as:

$$D = L\Delta R^T$$

and the rank-$r$ approximation $D_r$ is defined as:

$$D_r = L_r \Delta_r R_r^T = \sum_{i=1}^{r} l_i \delta_i r_i^T.$$

The first step for data imputation using SVD is to initialise all missing values in $D$ with either 0 or the mean of each column in $D$ (mean imputation). In a second step we iteratively perform a SVD of the *mean-(zero-) imputed* data matrix $D$. Therefore, we compute a rank-$r$ approximation $D_r$ and update the indices where the missing values have been with those from the rank-$r$ approximation. This procedure we repeat until the imputed values are below a certain tolerance threshold or a maximum number of iterations is reached (see implemented SVD imputation method in `impute.py`). In the last tutorial we tried out different values for $r$ until we found a $r$ value that led to an acceptable approximation

by visual inspection of an image. However, often we do not use images or we do not have the original data for comparison. Here we need a way to automatically find the optimal rank $r$.
The task of this homework will be to find the optimal rank $r$ for imputing a data matrix with 60% missing data entries. You can use the already implemented SVD imputation method `svd_imputation()` (in the file `impute.py`). For this exercise you have to implement the function `svd_imputation_optimised()` in `impute.py` in order to perform the following experiments:

(a) First you have to implement the function `svd_imputation_optimised()` in the file `impute.py`. In this function you will implement the procedure to find the optimal rank $r$ for imputing the missing values of an image $X_{missing}$. For this purpose, you have to generate a train and test set by using 70% of all non-missing values for training and the remaining 30% for testing. First, you have to generate a new training matrix $X_t$ in which you set all testing indices (30% of all entries) to nan. You then iteratively test different $r$ values from $1, \ldots, 30$ by imputing the training matrix and computing the mean squared error between the imputed testing points and the true testing points. After you found the rank $r$ with the minimum mean squared error you use this rank to impute the original data matrix $X_{missing}$. Show the result of your imputation by plotting the original image and the imputed image in your solution sheet.

(b) Show a plot for all tested ranks $r$ with the corresponding mean squared errors and highlight the optimal rank in the plot. What is the optimal rank $r$ and the corresponding minimum mean squared error?

(c) What would you observe if you choose a too large or too small value for the rank $r$?

## Exercise 2: Kernel PCA

In the following exercise you will compare the PCA implementation from the first Homework to a Kernel PCA. As dataset you will use artificial data of two half-moon shapes (`Moon`) where each half moon represents a different class (see. Figure 1). Therefore, you have to extend the `pca.py` file from the first homework with a variety of new functions (you can also use the `pca.py` file attached to this homework). In addition, you also have to modify the main file (`run_hw3.py`) in order to perform the following experiments:

(a) Perform a PCA as in Homework 1 on the `Moon` data using the implementations from Homework 1 (plot transformed data, highlight samples according to their class membership and show the results in your solution)! What can you observe when looking at the first two principal components?

(b) Next you will implement a Kernel PCA with a Gaussian radial basis function (RBF) kernel:

$$k(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|_2^2),$$

where $\gamma$ is a hyperparameter. For this purpose, implement the function `RBFKernelPCA()` in the file `pca.py` and return the transformed values using the first two principal components.

(c) Apply the `RBFKernelPCA` for the following $\gamma$ values $[1, 5, 10, 20]$ and compare the results (by plotting the transformed data for the first two principal components) to

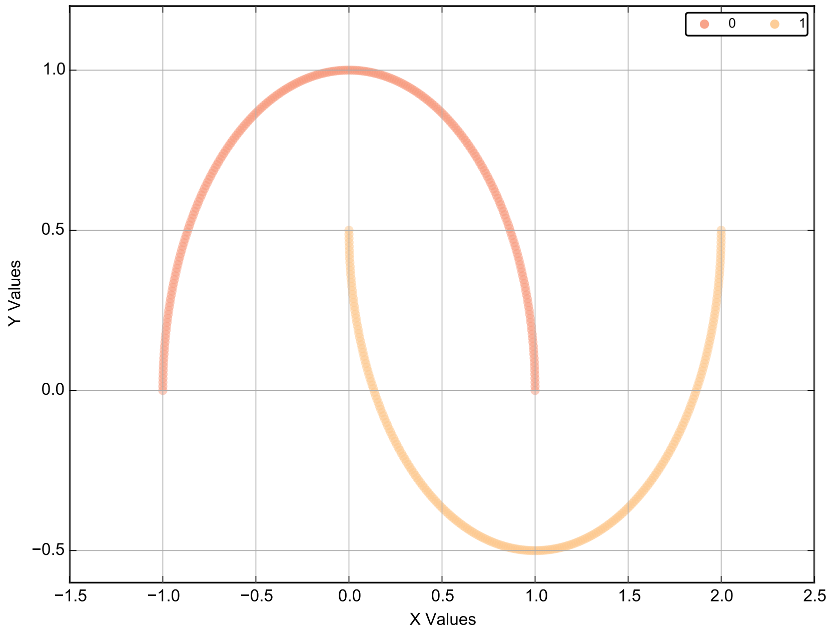

Figure 1: Non linear dataset of two half-moon shapes

those from (a). What do you observe? Could you think of a way how to find the optimal $\gamma$?

## Command-line arguments

Your program should generate all the plots and should output all necessary information in a readable format.
The file run_hw3.py must be executable using the following command:

```
$ python run_hw3.py
```

Note: **Zero** points for the **programming part** of this homework if your script is **not** executable with the above shown command line argument!
If you encounter package version problems, you may reuse the environment we used to compile these exercises (but are not forced to!). This can be either achieved using

```
$ poetry install
```

as called in the directory where pyproject.toml lies or via

```
$ pip install -r requirements.txt
```

using the provided requirements.txt file.

## Grading and submission guidelines

This homework is worth a total of 100 points. Table 1 shows the points assigned to each exercise/question. Follow the submission guidelines posted on the Moodle webpage. Refer to the document titled “General guidelines for homework sheets” (link named “General guidelines”).

Table 1: Grading key for homework 3

| | | |
|---|---|---|
| **60 pts.** | **Exercise 1** | |
| | 40 pts. | Exercise 1a |
| | 5 pts. | Exercise 1b |
| | 15 pts. | Exercise 1c |
| **40 pts.** | **Exercise 2** | |
| | 5 pts. | Exercise 2.a |
| | 25 pts. | Exercise 2.b |
| | 10 pts. | Exercise 2.c |

## Acknowledgments

This exercise was created by Karsten Borgwardt, Dean Bodenham, Dominik Grimm, Xiao He and Damian Roqueiro. Most recently, it was extended by Michael Moor.