Exercise 7

Deadline: 09.02.2022, 16:00.
Ask questions to #ask-your-tutor-felix

Regulations

Please hand in the following:

- PDF (hand written or typed) `proof-ridge-regression.pdf` for question 1.
- Jupyter notebook and exported HTML `kernelized-ridge-regression.*` for question 2.
- Jupyter notebook and exported HTML `fitting-circles.*` for question 3.

Zip all files into a single archive `ex07.zip` and upload this file to your assigned tutor on MaMPF before the given deadline.
Note: Each team creates only a single upload, and all team members must *join* it as described in the MaMPF documentation at `https://mampf.blog/zettelabgaben-fur-studierende/`.
Important: Make sure that your MaMPF name is the same as your name on Muesli. We now identify submissions purely from the MaMPF name. If we are unable to identify your submission you will not receive points for the exercise!

1 Proof - Ridge Regression - Primal vs Dual (10 Points)

In the primal formulation, the ridge regression problem takes the following form:

$$\widehat{\beta} = \operatorname{argmin}_{\beta} \|\mathbf{y} - X\beta\|_2^2 + \tau \|\beta\|_2^2 , \tag{1}$$

where X is an $N \times D$ matrix, β is a D-dimensional vector and $\mathbf{y}$ is an N-dimensional vector. As you saw in the lecture, the optimal $\widehat{\beta}$ is given by

$$\widehat{\beta} = \left(X^T X + \tau I_D\right)^{-1} X^T \mathbf{y} \, . \tag{2}$$

Here I_D is the D-dimensional unit matrix. You also know that the dual formulation of the problem is given by

$$\widehat{\alpha} = \operatorname{argmax}_{\alpha} - \alpha^T \left(XX^T + \tau I_N\right) \alpha + 2\alpha^T \mathbf{y}, \tag{3}$$

with the solution for $\widehat{\alpha}$

$$\widehat{\alpha} = \left(XX^T + \tau I_N\right)^{-1} \mathbf{y} \, . \tag{4}$$

For each feasible α, a corresponding feasible β is given by:

$$\beta = X^T \alpha \, . \tag{5}$$

Prove that the optimal $\widehat{\beta}$ corresponds to the optimal $\widehat{\alpha}$ (and not to some other α):

$$\widehat{\beta} = X^T \widehat{\alpha} \, . \tag{6}$$

Hint: Start by proving the following lemma (e.g. using the SVD of X). It will be useful in your derivation:

$$\left(X^T X + \tau I_D\right)^{-1} X^T = X^T \left(XX^T + \tau I_N\right)^{-1} \tag{7}$$

2 Kernelized (Ridge) Regression (10 Points)

From the lecture you know that using the dual formulation of the ridge regression, we are able to introduce the **kernel trick** with kernel matrix G. New regressed values y_{new} are computed through

$$\mathbf{y}_{new} = \mathbf{y}^T \left(G + \tau I_N\right)^{-1} \kappa^T \ . \tag{8}$$

The elements of the kernel matrix are computed on the training set via the kernel function K

$$G_{i_1 i_2} = K(X_{i_1}, X_{i_2}), \tag{9}$$

and the weight vector $\alpha = \mathbf{y}^T \left(G + \tau I_N\right)^{-1}$ only needs to be computed once at the beginning. In contrast, the kernel vector κ has to be recomputed for each new instance X_{new} according to $\kappa_i = K(X_{\text{new}}, X_i)$.

In this task, you shall apply kernel ridge regression to reconstruct all missing pixels in the grayscale image `cc_90.png` (available under "external link" on MaMPF). The features X_i are the pixel coordinates, and the response y_i is the corresponding grayvalue. Pixels with grayvalue $= 0$ are considered missing and shall be replaced with their regressed values. Use a squared exponential kernel function

$$K(X_{i_1}, X_{i_2}) = \exp\left(-\frac{\|X_{i_1} - X_{i_2}\|^2}{2\sigma^2}\right) \tag{10}$$

If you are careless in structuring your program, it will be too slow to run. Therefore, keep to the following guidelines:

- Cut off the kernel function (i.e. set it to zero) at a sensibly large radius to get a sparse kernel matrix, and use a sparse array class.
- Do not loop over each cell in the kernel matrix, vectorize at least one dimension.
- Do not explicitly invert any matrices. In general, but especially for sparse matrices, solving $A\mathbf{x} = b$ for $\mathbf{x}$ is many times faster than computing $\mathbf{x} = A^{-1}\mathbf{b}$ explicitly. For example, you can use `scipy.sparse.linalg.spsolve`

Do the same experiment for the Nadaraya-Watson kernel regression (a heuristic simplification of kernel ridge regression, which does not require the expensive pre-computed weights α):

$$\mathbf{y}_{new} = \frac{\sum_i \mathbf{y}_i \kappa_i}{\sum_i \kappa_i} \tag{11}$$

Play with the σ of the squared exponential as well as the τ of the ridge regression and find the parameters that produce the visually best reconstructed image for both approaches. Comment on the quality of the two methods.

3 Fitting Circles

In this exercise we will have a closer look at fitting circles to data. The numpy-file `circles.npy` (available under "external link" on MaMPF) contains many pairs of x-y-coordinates, and can be loaded through `data = np.load("circles.npy")`. Visualize the data in a scatter plot to show that the points are arranged in the shape of several circles and circle segments. Pay attention that the axes are scaled identically when plotting the data, otherwise your circles will look like ellipses. How many circles or circle segments would you fit into the data as a human?

3.1 RANSAC (6 Points)

First implement the RANSAC-Algorithm for the fitting of circles:

- For a set number of times N, repeat the following:
  - Randomly choose 3 points and determine the circle passing throuhg all of them, parametrized though its radius and the coordinates of the center. For example, you can achieve this by foming and solving a system of three equations, one for each point.
  - Classify points that are closer than ϵ to the circle as inliers and count them.
  - If the inlier count for this circle is higher than for the best circle so far, save the current circle and its inliers as the new best.
- Fit further circles by deleting all inliers of the last fitted circle from the dataset and repeat the procedure.

Estimate the number of iterations N that is needed, as you learnt in the lecture.

Plot all fitted circles on top of original data and comment on the result. Is the result sensitive to the value of ϵ, the parameter that is used to determine the inliers?

Hint: For plotting circles, you can use the following methods:

```
circle = plt.Circle((cx, cy), radius=r, fill=False) # Create a circle
plt.gca().add_patch(circle) # Add it to the plot
```

In the next two sub-tasks, you will further improve the fits of your circles using two different methods.

If you were not able to implement the RANSAC, then either try to fit a circle manually in the data and get the inliers this way, or create data for a circle + noise and use it for the rest of the exercise.

3.2 Algebraic Distance (3 Points)

For each set of inliers you found with RANSAC, fit a circle by minimizing the algebraic distance:

$$\min_{c,r} \sum_i \left(||x_i - c||_2^2 - r^2\right)^2 \tag{12}$$

You can use `scipy.optimize.least_squares` (or another optimization library of your choice) as an implementation. Plot the refined circles on top of the data and comment on the results.

3.3 Levenberg-Marquardt (7 Points)

For each set of inliers you found with RANSAC, solve

$$\min_{c,r} \sum_i \left(||x_i - c||_2 - r\right)^2 \tag{13}$$

by adapting the Levenberg-Marquardt algorithm to this problem[1]. Derive the theoretical approach by hand, but feel free to use `scipy.optimize.least_squares` (or another optimization library of your choice) as an implementation.

For the theory, first solve for r and express it in terms of x, c, then calculate the derivative with respect to c. Again, plot the resulting circles and comment on what you find.

[1]Since the Levenberg-Marquardt algorithm was not covered in the lecture this time, you can watch last year's recordings at `https://mampf.mathi.uni-heidelberg.de/media/17998/play`, starting around 36'30

3.4 Comparison (4 Points)

It was mentioned in the lecture that the solution from (13) is more robust to outliers than solving (12). Try to show this experimentally.

Hint: You can approach this problem for example as follows: Pick one set of inliers, sample a subset of those and maybe add a couple of outliers. Then apply your procedures from 3.2 and 3.3 to this dataset.