Optimization and Algorithms
Part 3 of the Project

João Xavier
Cláudia Soares
Instituto Superior Técnico
November 2020

1 Dimensionality reduction with Multidimensional Scaling

The dimensionality reduction task. Real-world datasets usually count a large number of features, where it is not clear which features are really important to establish the properties of our data. The dimensionality of our data points is composed of *signal* that carries relevant information, and irrelevant *noise*.

To have a clear example of why this could be useful, we illustrate these concepts with the example of the naïve experimenter in [1]. The scientist in question is trying to discover the general motion law of an ideal spring-mass system. This system consists of a ball of mass m attached to a massless, frictionless spring, with spring constant k. The ball is released a small distance away from equilibrium and the displacement $x \in \mathbb{R}$ is a an oscilatory function of time. The scientist in question, being unaware of physics laws, captures the movement of the ball with three cameras, given we live in a 3D world. His knowledge of the problem does not allow him even to place the cameras in orthogonal positions. After running a sophisticated algorithm that returns positions of the ball for each image in the three image sequences, he plots all the identified positions for each screenshot, as exemplified in Figure 1.

The question here is *how to retrieve the function of x from this dataset?* The solution presented by Shlens [1], will not retrieve directly a sine wave, as knowledge of physics does, but it will allow the experimenter to express his dataset in one dimension.

In general, dimensionality reduction techniques are often used to project a data set onto a lower-dimensional space, for example two dimensional, for the purposes of visualization, or noise reduction. A linear technique for this aim is multidimensional scaling, or MDS [2]. It finds a low-dimensional projection of the data such as to preserve, as closely as possible, the pairwise distances between data points. In the case where the distances are Euclidean, this is a linear method, like the well-known Principal Component Analysis (PCA). The MDS concept can be extended to a wide variety of data types specified in terms of a similarity matrix, giving nonmetric MDS.

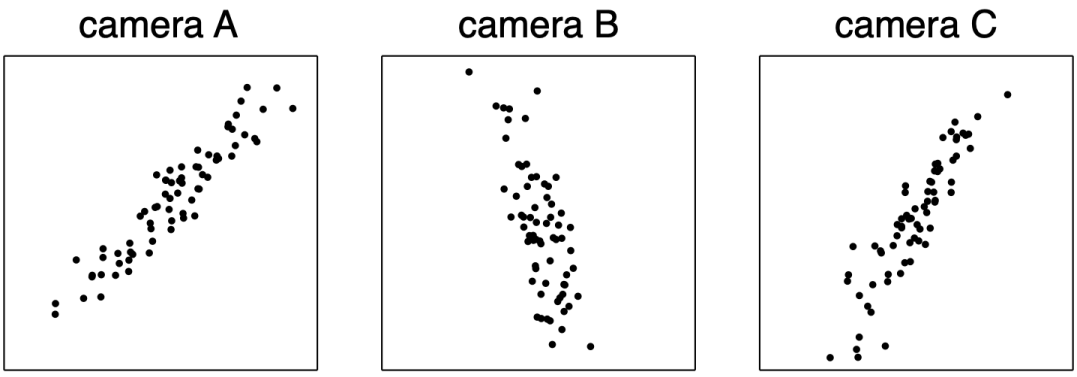

Figure 1: Positions of the ball for each image in the three image sequences. Figure from [1].

Our dataset is composed of N datapoints, $\{x_1, \ldots, x_N\}$, in a p-dimensional space, $x_n \in \mathbf{R}^p$, and we organize all datapoints in a data matrix

$$X = \begin{bmatrix} — & x_1^T & — \\ — & x_2^T & — \\ & \vdots & \\ — & x_N^T & — \end{bmatrix},$$

with N rows, one for each datapoint, and p columns, one for each feature.

To reduce the dimensionality p of our data vectors, we will compute all the distances between data points, resulting in a distance matrix D, such that each entry D_{mn} encodes the Euclidean distance between datapoints m and n, as

$$D_{mn} = \|x_m - x_n\|_2.$$

Obviously, the diagonal elements of D are all zero, and matrix D is symmetric, i.e., $D = D^T$, since $D_{mn} = D_{nm}$.

Task 1. Load the dataset in file `data_opt.csv` into Python or MATLAB. The file is in comma-separated values format, and you can load it in MATLAB, and in Python via Numpy or Pandas. Compute the distance matrix for the ten-dimensional dataset you loaded. Check that $D_{23} = 5.8749$, and $D_{45} = 24.3769$. Remember that if you are working in Julia or MATLAB the index of the datapoints starts in 1 while in Python it starts in 0. What is the largest distance you could find in your dataset? What pair of datapoints correspond to it?

In general, like the naïve experimenter, we do not know the true effective dimensionality of our data. So, we will try two different values for the target space dimensionality, $k \in \{2, 3\}$, and we will check the reconstruction error that we get with each one.

To perform MDS, we will try to match the distances in the high-dimensional original space of $\mathbf{R}^p$ to distances in the target space of $\mathbf{R}^k$, for two values of k.

The optimization problem. Denote the unknown positions on the low dimensional space as $y_n \in \mathbf{R}^k$, with $n = 1, \ldots, N$, and stack them in the vector variable

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}.$$

Finding the coordinates of the lower-dimension representation of our data can be posed as the following optimization problem:

$$\underset{y}{\text{minimize}} \quad \underbrace{\sum_{m}^{N} \sum_{n>m} \left(\|y_m - y_n\| - D_{mn} \right)^2}_{f(y)}. \tag{1}$$

This problem reflects the requirement that our low-dimensional representations will be as distant from each other as the high-dimensional corresponding pairs of datapoints.

1.1 Levenberg-Marquardt (LM) method

Consider the LM method given in slide 90 of the set of slides "Part 2: unconstrained optimization". The first step is to write problem (1) as

$$\underset{y}{\text{minimize}} \sum_{m=1}^{N} \sum_{n>m} (f_{nm}(y))^2.$$

To run the LM method you will need to compute the full $f(y)$, the full gradient $\nabla f(y) \in \mathbb{R}^{Nk}$, each function $f_{nm}(y) \in \mathbb{R}$ not yet squared, and each of the gradients of the $f_{nm}(y)$ functions, $\nabla f_{nm}(y) \in \mathbb{R}^{Nk}$. Note that the full gradient $\nabla f(y)$ *is not* the summation of the gradients $\nabla f_{nm}(y)$.

Example with three datapoints As an illustrative example consider $N = 3$, that we want to visualize in two dimensions. Our cost function will be

$$f(y) = (f_{12}(y))^2 + (f_{13}(y))^2 + (f_{23}(y))^2,$$

and the gradients $\nabla f(y)$, $\nabla f_{12}(y)$, $\nabla f_{13}(y)$ and $\nabla f_{23}(y)$ will lie on $\mathbb{R}^6$.

Task 2. Write the expressions for $f_{nm}(y)$, and analytically compute the expressions for the gradients $\nabla f(y)$, $\nabla f_{nm}(y)$. Write matrix A and vector b defined in slide 91 of the set of slides "Part 2: unconstrained optimization".

Now we have everything we need to obtain our low-dimensional representations.

Task 3. We will try both $k \in \{2, 3\}$, and see which one better fits our dataset.

Solve problem (1) using the LM method given in slide 90 of the set of slides "Part 2: unconstrained optimization," with parameters $\lambda_0 = 1$ and $\epsilon = k10^{-2}$. Initialize your method with the vector stored in file `yinit_k.csv`, where k will take each of the two values 2 and 3.

The dataset for problem (1) is stored in the file `data_opt.csv`, loaded in Task 1. This file contains a data matrix with 200 rows (number of datapoints), and 10 columns (collected features).

After running your code, plot the estimates of the the low dimension positions given by the LM method, the vaue of the cost, and the norm of the gradient of the cost function along iterations, for both values of k. (So you can check your code, we provide these answers in Figures 2-5, which you should reproduce.)

After the algorithm is setup you can apply it to any real-valued dataset. You will do that in the next task.

Task 4. Solve problem (1) using the LM method given in slide 90 of the set of slides "Part 2: unconstrained optimization," with parameters $\lambda_0 = 1$ and $\epsilon = k10^{-4}$. We now assume $k = 2$. The data for problem (1) is in the file `dataProj.csv`; for this exercise, we do not provide an initialization. You should run the LM method from several random initializations, thereby obtaining several solutions; report solutions with the smallest value of the cost function. What can you say of the uniqueness of the solution?

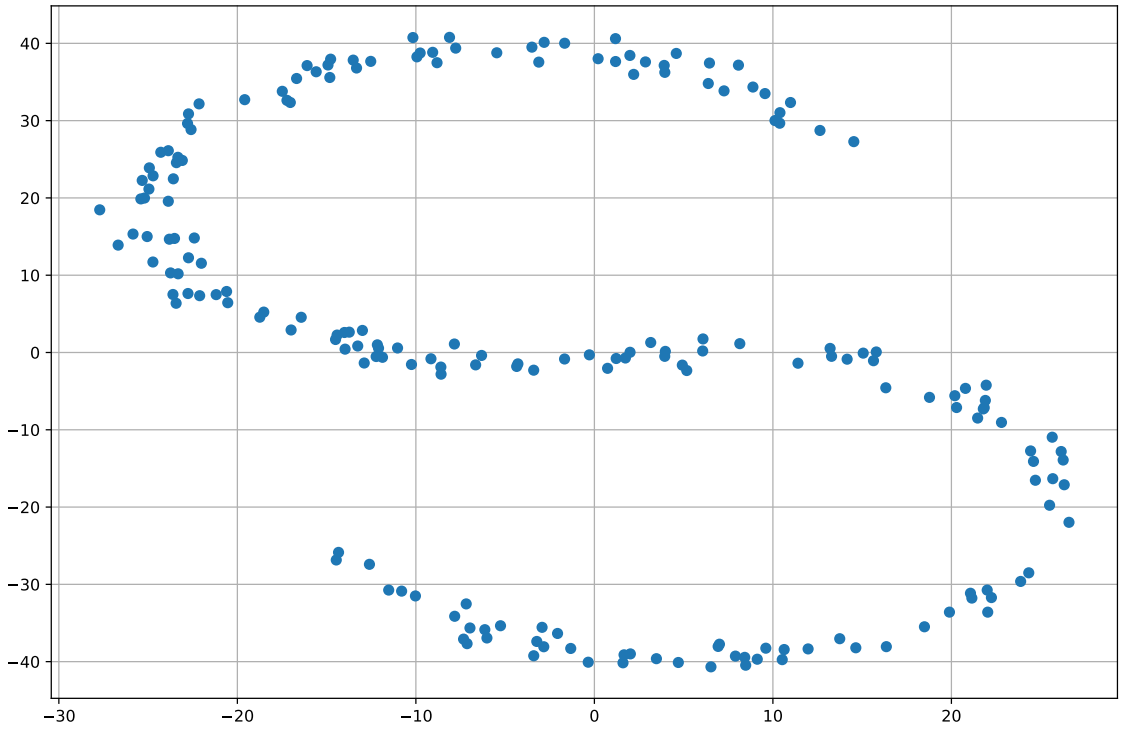

Figure 2: Dimensionality reduction output for $k = 2$.

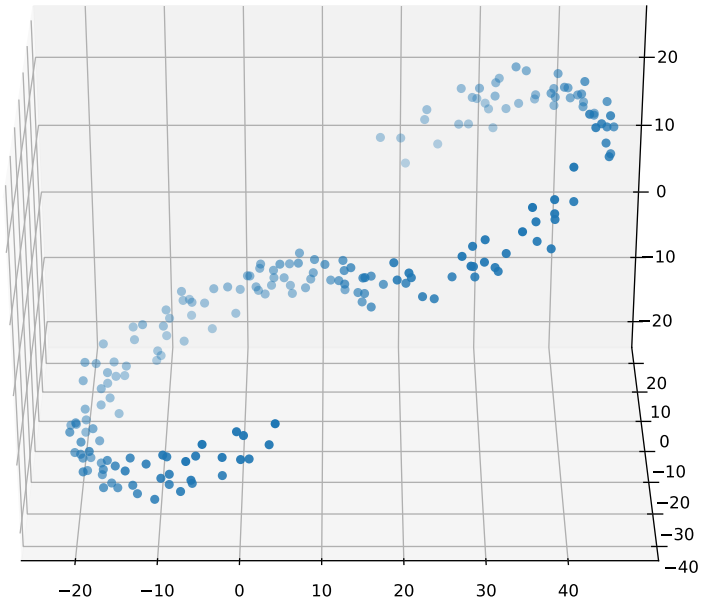

Figure 3: Dimensionality reduction output for $k = 3$.

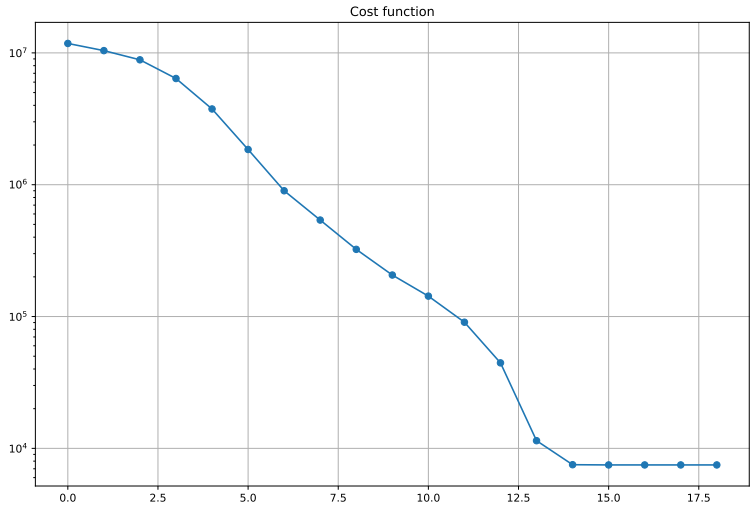

Figure 4: Cost value along iterations of the LM method, when the LM method is applied to problem (1) with the data of file `data_opt.csv` (Task 3, $k = 2$).

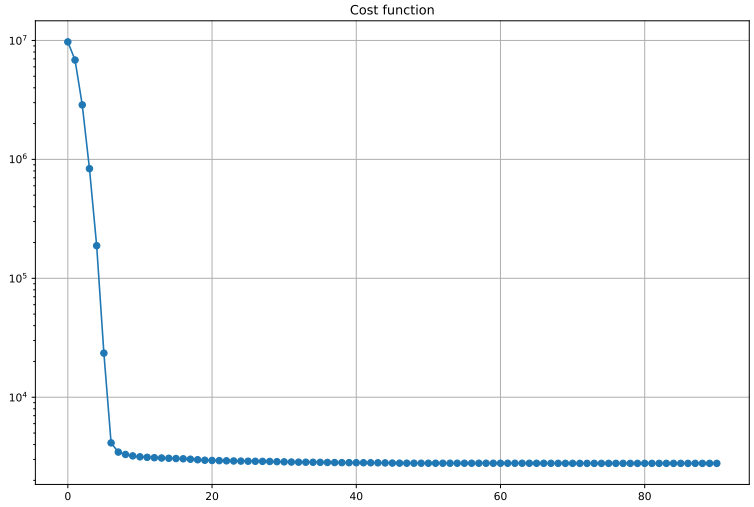

Figure 5: Cost value along iterations of the LM method, when the LM method is applied to problem (1) with the data of file `data_opt.csv` (Task 3, $k = 3$).