Numerical Project in Python

Cooperative Kernel regression

Andrea Simonetto

version: January 8, 2025

You will consider the machine learning problem of kernel regression. This is traditionally formulated as an optimization problem and you will code different ways to distributed the computation across a network of communicating agents. **The aim is to obtain the same graphs that I have presented in class and you can find in the lecture notes**.

This project counts 40% **of your grading.**

Some instructions.

- The project is done in groups of three, everybody needs to do a comparable amount of work.
- You need to submit a single .pdf file, of no more than 10 pages, that is readable (for example, label all your graphs in a readable manner, with a reasonable fontsize), and it has inside all the elements to assess your work. Name the file: `LastName1_LastName2_LastName3_Final.pdf`
- In order to help you generate readable graphs, you have access to a setup example python script. It saves a sample figure in .pdf, and so must you. Fail to generate readable graphs will result in a **-3 points**.
- You also need to submit a python script that I can run to assess the correctness of your plots and graphs. The script will have to generate the results that you have in the report. And it has to be readable. Name the file: `LastName1_LastName2_LastName3_Final.py`
- The deadline for sending us the **TWO** files is Tuesday April the 2nd, at 13H00 Paris time, via the Moodle page of the course. **No possible extensions. No project = grade of zero for this part, no resit**.
- As you figured already, you will use python, and in particular you will need the following packages (please refrain from using any other packages that are not strictly needed).

```
import numpy as np
import matplotlib.pyplot as plt
import pickle
```

- Do as much as you can during the TP sessions at ENSTA, but it is possible that you will need to do extra work in addition to it.
- Don't wait the last day to put together the project, **it's quite some work!**

Getting started

Kernel ridge regression is a machine learning task that amounts to fit a functional model to noisy data, in a non-parametric form. You can think of it as linear regression plus plus.

Let y_i for $i = 1, \dots, n$ be scalar evaluation of a certain unknown function $f(x) : \mathbf{R} \to \mathbf{R}$ at points x_i for $i = 1, \dots, n$. In our case,

$$y_i = f(x_i) + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, \sigma^2).$$

We use a kernel representation of the function as

$$f(x) = \sum_{i=1}^{n} \alpha_i k(x, x_i),$$

where $k(x, x_i)$ is the kernel. Here we will use an Euclidean kernel as

$$k(x, x_i) = \exp(-\|x - x_i\|^2).$$

We also define the kernel matrix as $K = [k(x_i, x_j)]_{i,j=1,\ldots,n}$.

Function f is linear in the parameters α_i. To lower the computational requirement, we also use a Nyström approximation. We select uniformly at random amongst the n points, $m \sim \sqrt{n}$ points. We let $\mathcal{M}$ be the set of indexes of these points w.r.t. the original set of points. For example, if $m = 3$ then $\mathcal{M}$ could be the set $\{1, 4, 7\}$, corresponding to the first, fourth, and seventh point of the original n points.

The approximation then reads

$$f(x) = \sum_{i=1}^{n} \alpha_i k(x, x_i) \approx \sum_{j \in \mathcal{M}} \alpha_j k(x, x_j).$$

Determining the vector $\alpha \in \mathbf{R}^m$ is a model training problem which can be written as,

$$\alpha^* = \arg\min_{\alpha \in \mathbf{R}^m} \frac{\sigma^2}{2} \alpha^\top K_{mm} \alpha + \frac{1}{2} \|y - K_{nm}\alpha\|_2^2,$$

where $K_{nm} = [k(x_i, x_j)]_{i=1,\ldots,n; j \in \mathcal{M}}$, and y is the stacked version of all y_i.

Note then that,

$$\|y - K_{nm}\alpha\|_2^2 = \sum_{i=1}^{n} \|y_i - [k(x_i, x_j)]_{j \in \mathcal{M}} \alpha\|_2^2.$$

To make things simpler, we also add a small regularisation, such that,

$$\alpha^* = \arg\min_{\alpha \in \mathbf{R}^m} \frac{\sigma^2}{2} \alpha^\top K_{mm} \alpha + \frac{1}{2} \|y - K_{nm}\alpha\|_2^2 + \frac{\nu}{2} \|\alpha\|_2^2,$$

for $\nu = 1.0$, and the problem becomes strongly convex and smooth.

1 Part I (Classes 1 and 2)

- Download the data file, which is a file containing the x_i points (or features), and y_i noisy data (or labels). Here we will use $\sigma = 0.5$. There are one million points.

```
with open('first_database.pkl', 'rb') as f:
   x,y = pickle.load(f)
```

- Visualize the data.
- Choose the first $n = 100, m = 10$ points and share them across $a = 5$ agents or computers. The problem reads,

$$\alpha^* = \arg\min_{\alpha \in \mathbf{R}^m} \sum_{a=1}^{5} \left[\frac{\sigma^2}{5} \frac{1}{2} \alpha^\top K_{mm} \alpha + \frac{1}{2} \sum_{i \in A} \|y_i - K_{(i)m}\alpha\|_2^2 + \frac{\nu}{10} \|\alpha\|_2^2 \right],$$

  where we indicated with $i \in A$ the data points that belong to agent A, and with $K_{(i)m} = [k(x_i, x_j)]_{j \in \mathcal{M}}$.

  To select the points, it may be useful to use,

```
sel = [i for i in range(n)]
ind = np.random.choice(sel, m, replace=False)
x_selected = [x[i] for i in ind]
```

- Let each agent have 20 points, and let each agent have different points. Set a communication graph (connected and undirected) between the agents and solve the problem above with

  1. Decentralized gradient descent
  2. Gradient tracking
  3. Dual decomposition
  4. ADMM

In all the cases, justify your choices of step size from theory. Plot the optimality gap ($\|\alpha_t^i - \alpha^*\|$) as the number of iterations t increases (in a log-log plot). You can compute the true α^* by solving the problem in a centralized fashion as a linear algebra problem.

- Plot the convergence varying the graph structure (from a line to a small-world graph to a fully connected one).
- Visualize the obtained functions by testing them on a uniform grid of dimension $nt = 250$, that is for the points x' computed as,

  ```
  x_prime=np.linspace(-1,1,nt)
  ```

- Try to break convergence by adding directed communication, package losses, asynchronicity.
- (Optional) Recover convergence with the push-sum protocol in the case of directed communication.
- In all the above please justify what you see by the theory that you have studied in class.
- Increase n as much as you can, maintaining $m = \lceil \sqrt{n} \rceil$, and redo the above. Here you are also allowed to increase the number of agents if needed. In this case, the centralized solution may not be computable, so then plot other metrics to gauge convergence.

  The team that has reached the highest n gets one bonus point.
- How convergence depends on n?

2 Part II (Classes 3 and 4)

Consider back the case of $n = 100$, $m = 10$, and $a = 5$. Consider the second database and assign each groups of 20 points to the different agents.

```
with open('second_database.pkl', 'rb') as f:
   X, Y = pickle.load(f)
```

In particular `X[i]` and `Y[i]` correspond to the data available to agent i.

As you may have noticed, the points m do not have to be necessarily points for which we have labels. Here it is convenient to use

```
x_m_points=np.linspace(-1,1,m)
```

- Implement FedAvg, for $B = 20$ $C = 5$, and $E = 1, 5, 50$ epochs with constant learning rate.
- Display the convergence of the global model in terms of objective error.
- Consider different parameters for the number of batches, epochs, and selected clients and show some plots showcasing the different behaviour. Here you can also use a diminishing learning rate.
- Comment the plots and results in relation to what you have studied in theory.
- (Optional) The data in this database are not necessarily IID across the groups of 20 points. Implement SCAFFOLD to fix it.

3 Part III (Class 5)

Consider again the problem of Part I (the first database) and consider the algorithm DGD-DP. We want now to implement the latter to solve our regression problem in a distributed and DP way.

- Consider $n = 100$, $m = 10$, and $a = 5$, and plot optimality gaps (e.g., $\|\alpha_t^i - \alpha^*\|$) as the number of iteration increases, for your choice of step size, noise, and other parameters. Referring to [arXiv:2202.01113] you can choose to select the parameters as in their Theorem 2, part 3) to obtain a DP of $\epsilon = 0.1, 1, 10$, with the other parameters fixed as in class.
- (Optional) Consider now $n = 1000$, $m = 33$, and $a = 100$ and redo the above. Comment what you observe.