# Numerical Project in Python
Cooperative Kernel regression

Andrea Simonetto

version: January 9, 2023

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.

Some instructions.

- You can do the project in pairs, as long as everybody does a comparable amount of work.
- You need to submit a single .pdf file, with both projects, 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_Final.pdf`
- You also need to submit a jupyter notebook that we can run to assess the correctness of your plots and graphs. The notebook will have to generate the results that you have in the report. And it has to be readable. Name the file: `LastName1_LastName2_Final.ipynb`
- The deadline for sending us the **TWO** files is Tuesday April the 4th, at 12H00 Paris time, via the Moodle page of the course.
- 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
import cvx
```

- 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.

## 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 $K = [k(x_i, x_j)]_{i,j=1,\dots,n}$.

Function $f$ is linear in the parameters $\alpha_i$. 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,

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

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,

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

## 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,

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

  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]
```

- 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 (is the function strongly convex and smooth?). 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 $\alpha^*$ by solving the problem in a centralized fashion in CVX, or 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 $s = 250$, that is for the points $x'$ computed as,

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

- 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 $B = 10$ batches, $E = 10$ epochs, and selecting at each epochs $C = 3$ clients.
- 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.
- 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 stepsize, 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 $\epsilon = 0.01$ for a $C = 1$.
- (Optional) Consider now $n = 1000$, $m = 33$, and $a = 100$ and redo the above. Comment what you observe.

## 4 Part IV: optional

In some case, you want to be able to impose functional constraint on the function that you are reconstructing. This amounts at adding constraints to the regression problem.

A typical constraint is convexity. For instance you want to find convex functions that fit the data as close as possible.

A way to do this is to impose convexity on sampled points. Let $S$ be a set of points $x$ on which you want to impose convexity. Remember that,

$$f(x) = \sum_{j \in \mathcal{M}} \alpha_j k(x, x_j), \qquad \nabla_x f(x) = \sum_{j \in \mathcal{M}} \alpha_j \nabla_x k(x, x_j).$$

Let also be $f_i = f(x_i)$ and $g_i = \nabla_x f(x_i)$, for our case, both scalar.
Then the convex constraint read:

$$f_i - f_j - g_j(x_i - x_j) \geq 0 \qquad \forall i, j \in S.$$

The above can be compactly written as a linear constraint: $A\alpha - b \leq 0$, comprising of $s(s-1)/2$ scalar constraints (if $S$ contains $s$ points).
So we want to solve,

$$\begin{aligned} \alpha^* &= \arg \min_{\alpha \in \mathbf{R}^m} \frac{1}{2} \alpha^\top K_{mm} \alpha + \frac{1}{2\sigma^2} \|y - K_{nm}\alpha\|_2^2, \\ & \quad \text{subject to } A\alpha - b \leq 0. \end{aligned}$$

- Propose ways to solve the above in a distributed way by using either dual decomposition or ADMM.
- Implement the proposal.