GenHack3 - Simulation of the maize yield across the years in the context of the climate change

January 22, 2024

1 Introduction

1.1 Context

Agriculture's exposure to climate impacts is not new, and farmers are the first witnesses and victims of these impacts. Increasingly early, intense and frequent heatwaves, changes in rainfall patterns and late frosts are having an increasingly harsh effect on agricultural production. Given the importance of this sector, there is an urgent need to define resilient solutions for the future climate and develop new tool. Changes in the climate can be seen in episodes of heavy rainfall, drought, and temperatures which can undermine food security by slowing down growth on a global scale. This drought is specifically affecting many crops, particularly maize. Maize is the world's most important cereal crop, making it an essential resource for millions of people.

⭐ **Objective**
The purpose of this challenge is to build a spatial generator of annual maize yield.

2 Dataset description

2.1 The variables

We consider $S = 4$ maize field (stations) located in French department $(49, 80, 40, 63)$, see Figure 1. We model the annual maize yield as a random vector $Y = (Y_1, Y_2, Y_3, Y_4)$ in dimension $S = 4$ where from now on, we use the following indexing for the stations:

1 : **station 49**, 2 : **station 80**, 3 : **station 40**, 4 : **station 63**.

We also consider weather variable $W = (W_1, W_2, W_3, W_4)$ in dimension D of both the temperature and the rainfall (2 variables) at each station during 9 periods, *i.e.* $D = 2 \times 4 \times 9 = 72$. The weather data is observed from April 27 to October 27 (9 periods), this corresponds to the time of the year when maize is cultivated. The periods are divided as follows

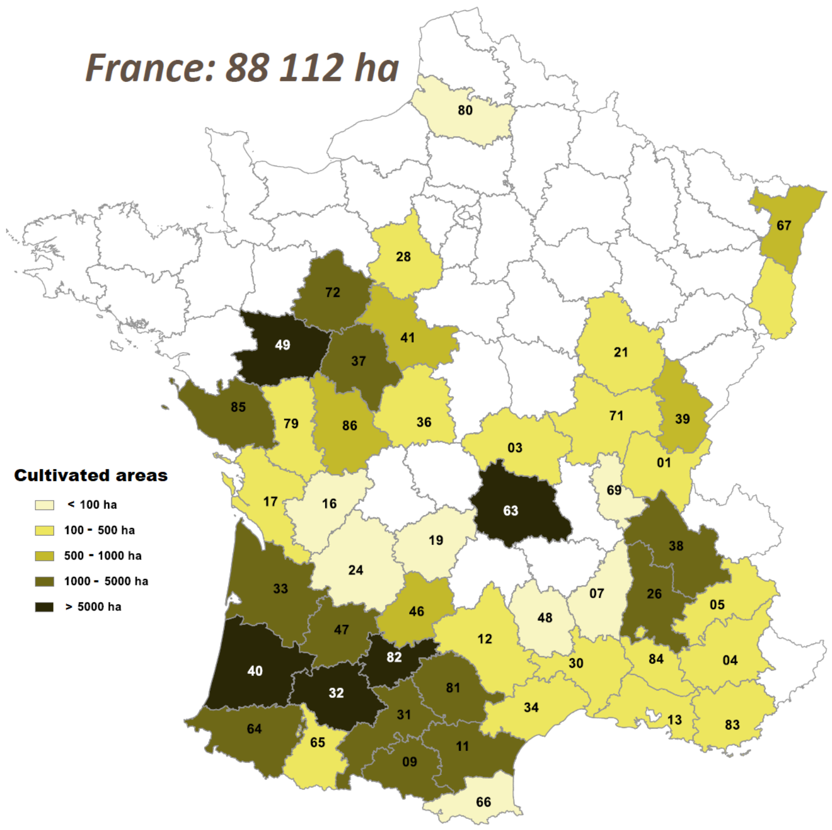

Figure 1: Maize cultivated area in France per department (county) in 2022. Each color corresponds to the cultivated surface. The number on each department correspond to the zip code, e.g., 63 is *Puy-de-Dôme*.

- Period 1: April 27 - May 16,
- Period 2: May 17 - June 5,
- Period 3: June 6 - June 25,
- Period 4: June 26 - July 15,
- Period 5: July 16 - August 4,
- Period 6: August 5 - August 24,
- Period 7: August 25 - September 13,
- Period 8: September 14 - October 3,
- Period 9: October 4 - October 27.

For each station $j \in \{1, \ldots, S\}$, the weather variable W_j is defined as

$$W_j := \begin{bmatrix} W_j^{(1)} \\ \vdots \\ W_j^{(9)} \\ W_j^{(10)} \\ \vdots \\ W_j^{(18)} \end{bmatrix} = \begin{bmatrix} \texttt{average}\text{ daily maximum temperature - period 1 - station j} \\ \vdots \\ \texttt{average}\text{ daily maximum temperature - period 9 - station j} \\ \texttt{average}\text{ daily rainfall - period 1 - station j} \\ \vdots \\ \texttt{average}\text{ daily rainfall - period 9 - station j} \end{bmatrix}.$$

The daily maximum temperature (in degree Celsius °C) is the maximum temperature recorded during a day. The daily rainfall is the total accumulated rain amount in mm/m^2.

Did you know?
Maize is more or less sensitive to weather stress, depending on the period.

2.2 The observations & Hypothesis

Let $\{y_i \in \mathbb{R}^S, w_i \in \mathbb{R}^D\}$ be the couple of observations at the i-th year with $i \in \{1, \ldots, n_{\text{train}}\}$.

- We assume that each year the $\{y_i, w_i\}$ are independent and identically distributed (i.i.d.).
- The weather variables across the different station are **NOT** independent.
- The yields are conditionally independent, i.e. $Y_i \mid W_i = w_i$ is independent of $Y_j \mid W_j = w_j$ for $i \neq j$.
- Furthermore, the yield at each station has the same marginal (unconditional) distribution, i.e. Y_i has the same distribution for all $i = \{1, \cdots, S\}$.

Important!

- The exact position of the station will not be disclosed during the challenge.
- The way the yield and weather dataset were obtained will not be discussed during the challenge.

3 The task: 1st round

You will have to build a generator of the form $Z \mapsto G(Z)$ in dimension $S = 4$ where Z is the input noise (latent variable) in dimension d_z. The latent variable Z will be a **standard normal random vector**, (*i.e.* all the components are independent and zero-mean unit-variance normally distributed random variable).

In the first round, you will have to build this generator G only for a subset of the training data. This subset is defined by

$$\mathcal{E} = \Big\{W_1^{(13)} + W_1^{(14)} + W_1^{(15)} \leq Q_1, W_2^{(13)} + W_2^{(14)} + W_2^{(15)} \leq Q_2,$$
$$W_3^{(13)} + W_3^{(14)} + W_3^{(15)} \leq Q_3, W_4^{(13)} + W_4^{(14)} + W_4^{(15)} \leq Q_4\Big\}$$

where

$$Q = \begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \end{bmatrix} = \begin{bmatrix} 3.3241 \\ 5.1292 \\ 6.4897 \\ 7.1301 \end{bmatrix}$$

In words, it corresponds to a French summer (June 26 to August 24 i.e. period 4,5 and 6) where the cumulated average rainfall at each station is **simultaneously** below some low value corresponding approximately to the 43rd percentile at each station.

$\triangleright$ The goal is to make the distribution of $G(Z)$ as close as possible to the distribution of the yield on this subset $\mathcal{E}$.

Warning!

The subset $\mathcal{E}$ of interest satisfies the four conditions simultaneously, i.e. dry summer all over the four stations. This type of joint "extreme" are called in climate science *compound events*. We are interested in reproducing the yield joint vector Y on this subset and not only its marginals.

In short, the first task is a "classic" generative problem on a 4-dimensional distribution on i.i.d. data. The originality comes from the fact that we focus on a specific subset $\mathcal{E}$ of the training data (it does not mean that you cannot use in some ways the remaining data).

3.1 What you need to deliver

At the first evaluation, we will evaluate your model

$$Z = \begin{bmatrix} Z^{(1)} \\ \vdots \\ Z^{(d_z)} \end{bmatrix} \mapsto G_\theta(Z)$$

where G_θ is your generative model parameterized by θ is a mandatory output structure

$$G_\theta\left(\begin{bmatrix} Z^{(1)} \\ \vdots \\ Z^{(d_Z)} \end{bmatrix}\right) = \begin{bmatrix} \widetilde{Y}^{(1)} \\ \vdots \\ \widetilde{Y}^{(S)} \end{bmatrix},$$

with a common and unknown random vectors Z with $d_z \leq 50$ (**the latent dimension is free as long as it is less than** 50).

Danger zone!

Your generative model must be a **non-constant** function of Z! Otherwise, it will return a Delta distribution! We will use your generator G_θ with our Z to generate your result $\widetilde{Y}$.

3.2 Train-test dataset

We provide a training period of 10,000 years (n_{train} =10,000), while the testing period is 100,000 years (n_{test} =100,000).

Teams will be judged on the test set based on one evaluation score metric: the Slice Wasserstein Distance (see next Section). **We** will use independent latent variables $Z_1, Z_2, \ldots, Z_{n_{\text{eval}}}$ as input noise to generate the appropriate number of testing data for each evaluation, where n_{eval} is the number of evaluation points.
▷ For the first evaluation (1st round), n_{eval} =9,941.

4 The task: 2nd round

To be announced. It will be spicier ⭐⭐⭐.

5 Evaluation

5.1 Evaluation of the Wasserstein Distance

The criterion for evaluation is the *Sliced Wasserstein Distance* (SWD), which is a computationally tractable approximation of the *Wasserstein Distance* (WD). WD is a mathematical notion that gives a measure of the distance between two distributions, think for example of two clouds of data points. Mathematical background is given in the appendix at the end of this document.
In this challenge, we will use SWD to measure the distance between the empirical distribution of your generator G_θ with a test dataset.

SWD implementation for evaluation. The used implementation of the SWD is the function '`ot.sliced.sliced_wasserstein_distance`' of python package POT[1] In the package documentation, you will find simple examples on how to use it as well as the details of the implementation.

On our side: for the evaluation, we sample a large enough set of projection angles from a determined seed so that the computation of the SWD is precise and deterministic; no statistical fluctuation is to be introduced to any source of randomness in the computation of the SWD, in the interest of fairness of evaluation.

Tips
We recommend that you incorporate this metric (among others) to test/validate your training.

5.2 Ranking

At each evaluation, the ranking will be determined by

1. Computing a Z-score $\mathcal{Z} = (x - \mu)/\sigma$, where x is the result of the SWD of your generator, μ and σ the average and standard deviation of SWD over all teams.

[1] `https://pythonot.github.io`

2. Converting to points $P = (1 - \Phi(\mathcal{Z})) * 1000$, with Φ the c.d.f. of a standard normal distribution

6 General rules

- All your code must be in Python 3.11.
- Git will be used to push your model and Docker to build an environment to evaluate your code. You don't have to master git and Docker, just have them installed in your local machine.
- No restriction on how you train your model, but before each submission date you must ensure that your model runs in our predefined framework (see the Paragraph Submission below).

Requirements.

- Python 3.11
- Git
- Docker

Create your private repository.

1. Visit the Genhack3 git repo where you will find:
   - `data/`: folder containing the training data (`station_49.csv`, `station_80.csv`, `station_40.csv`, `station_63.csv`) and an example of a noise file (`noise.npy`). Feel free to use another noise for training your model but keep in mind that
     - d_z must be less or equal to 50,
     - you will be evaluated on a common (to all participants) and unknown standard normal random vector Z.
   - `requirements.txt`: text file containing the libraries with their associated versions you used in the `model.py` file. **Do modify ✓**
   - `Dockerfile`: docker image in order to create a container. **Do not modify ✗**
   - `main.py`: main python file containing the simulation function. **Do not modify ✗**
   - `model.py`: python file containing your generative model and for loading the parameters. **Do modify ✓**
   - `parameters/`: folder where you must put the parameters of your model. **Do modify ✓**
   - `run.sh`: bash script to run your simulation. **Do not modify ✗**
2. Duplicate the repository (just one of the team member)
   (a) create an empty **private** repository with as repo name the **NAME_OF_YOUR_TEAM**. Be sure that your 'base' branch is called `master`.
   (b) invite your team members and **generative-hackathon**

(c) open a terminal where you want to put your private repo and run the following commands.
Note: you can either clone on HTTPS (default) or on SSH depending on your setting.

```
$ git clone --bare
    https://github.com/generative-hackathon/Genhack3.git (HTTPS)
OR
git clone --bare git@github.com:generative-hackathon/Genhack3.git (SSH)
$ cd Genhack3.git
$ git push --mirror
    https://github.com/YOUR_GIT_USERNAME/NAME_OF_YOUR_TEAM.git (HTTPS)
OR
git push --mirror
    git@github.com:YOUR_GIT_USERNAME/NAME_OF_YOUR_TEAM.git
$ cd ..
$ rm -rf Genhack3.git
$ git clone https://github.com/YOUR_GIT_USERNAME/NAME_OF_YOUR_TEAM.git
    (HTTPS)
OR
git clone git@github.com:YOUR_GIT_USERNAME/NAME_OF_YOUR_TEAM.git
```

Submission.

1. open Docker
2. put in your local repo your latest version of `models.py` and `parameters/` and push them to your git repo
3. open a terminal where your local repo is, and run the command

```
$ sh run.sh
```

   This command will

   (a) pull your git repo - so again, be sure that your git repo is up to date
   (b) run your code in a docker container
   (c) push to your git repo a `check.log` file containing a debug message

4. open the file `check.log` to see if you had a ”successful simulation”. If not, fix the error and run again

```
$ sh run.sh
```

7 Schedule

- Evaluation #1: January 31, 2024 - 11:59 Paris time
- Evaluation #2: February 10, 2024 - 11:59 Paris time

References

[1] Kimia Nadjahi. *Sliced-Wasserstein distance for large-scale machine learning : theory, methodology and extensions.* Theses, Institut Polytechnique de Paris, November 2021.

A Mathematical background on the Wasserstein Distance and the Sliced Wasserstein Distance

Wasserstein Distance. The Wasserstein Distance is a distance on the space of measures. It effectively measures the discrepancy between two random variables.
Consider two measures μ, ν on $\mathbb{R}^d$. Considering a cost function $c : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$ and $p \geq 1$, the p -Wasserstein distance associated to cost c between μ and ν is defined as:

$$W_p(\mu, \nu) = \left(\inf_{\gamma \in \Gamma(\mu,\nu)} \mathbb{E}_{(x,y)\sim\gamma} \left[c(x,y)^p \right] \right)^{1/p},$$

where $\Gamma(\mu, \nu)$ is the set of *couplings* of μ and ν, *i.e.* the set of distributions on $\mathbb{R}^d \times \mathbb{R}^d$ such that the marginals are respectively μ and ν. For dimension 1, there exists a closed formula of the WD for empirical measures. Consider samples $X = \{X_1, \ldots, X_n\}$, $Y = \{Y_1, \ldots, Y_n\}$ two dataset of real valued points, and their ordered versions $X = \left\{X_{(1)} \leq \cdots \leq X_{(n)}\right\}$, $Y = \left\{Y_{(1)} \leq \cdots \leq Y_{(n)}\right\}$:

$$W_p(X, Y) = \left(\frac{1}{n} \sum_{i=1}^{n} c\left(X_{(i)}, Y_{(i)}\right)^p \right)^{1/p}. \tag{1}$$

However, for dimensions higher than 1, there exists no general formula.
The Wasserstein distance finds a nice interpretation in the theory of optimal transport. For curious readers, we suggest the high-quality Wikipedia page on the matter[2] and references therein.

Sliced-Wasserstein Distance (SWD). The Wasserstein Distance suffers from the fact that there exists no closed formula to compute it, nor very accessible computable estimates in dimensions higher that 1. In order to perform estimation, we therefore rely on the Sliced-Wasserstein Distance [1], which is an efficient proxy, much used for computations in experimental settings. It is computed by randomly projecting the two compared measures on axis defined by angles θ and averaging over the projections. With the same notations as in the previous paragraph, the Sliced-Wasserstein Distance is defined as:

$$\mathcal{SWD}_p(\mu, \nu) = \mathop{\mathbb{E}}_{\theta \sim \mathcal{U}\left(\mathbb{S}^{d-1}\right)} \left(\mathcal{W}_p^p \left(\theta_{\#}\mu, \theta_{\#}\nu \right) \right)^{1/p}, \tag{2}$$

where $\theta_{\#}\mu$ (respectively ν) stands for the pushforward of the projection $X \in \mathbb{R}^d \mapsto \langle X, \theta \rangle \in \mathbb{R}$, where $X \in \mathbb{R}^d \sim \mu$ (respectively ν).
As opposed to the WD, the SWD is an open door easy computable estimates, as it is an averaging of 1-dimensional WDs for which there exists a closed formula, formula (1).
Computationally, the empirical version of the SWD is based on the Monte Carlo estimator of equation (2).

[2] `https://en.wikipedia.org/wiki/Wasserstein_metric`