Report Defi IA 2022 : Predict the accumulated daily rainfall on ground stations

8 group : INSA - MA - Crying in the Rain
Jérôme Deveaux
Folke Skrunes
Aleksander Stangeland

Supervisor :
David Bertoin

14th January, 2022

1 Introduction

As part of the course AI Frameworks, our group participated in the Defi-IA 2022 competition hosted on Kaggle. This challenge consists in predicting the accumulated daily rainfall observed by weather stations. The data is provided by MeteoNet between 2016 and 2019. The goal is to construct a model based on training data from 2016 to 2017 and evaluate on test data from 2018 to 2020.

After a pre-processing stage of the data, we will apply various methods seen in the course to get the best possible predictions of daily rainfall. Our goal after preparing the data is to train a initial predictive model. Then we will try to improve our prediction using feature engineering, parameter tuning and testing different models.

2 Pre-processing of data

2.1 Data Processing

To retrieve all the Defi-IA data we needed, we collected them in dataframes, one for training and one for testing. The files we retrieved are the following :

- `X_station_train.csv` and `X_station_test.csv` : contains ground station observations (land measurements of 6 weather parameters by meteorological stations (e.g. wind, temperature humidity, rainfall)).
- `Baseline_forecast_train.csv`,`Baseline_observation_train.csv`, `Baseline_forecast_test.csv` and `Baseline_observation_test.csv` : contains forecasts from METEO FRANCE based on a physical model of the atmosphere for train data and test data.
- `stations_coordinates.csv` : contains ground station coordinates and elevation.

We reshaped and processed these tables in order to use them for models. While the baseline and ground truth data were on a per-station-per-day basis, the rows of the ground station observation data were per-station-per-hour. In order to fix this, we pivoted the `X_station` columns so each row corresponded to a station and a day, and the columns were all the observations for that day put side by side. Pivoting the data in this way, we obtained a table with 144 variables, corresponding to the 6 observed variables for the 24 hours of each day.

2.2 Feature engineering

In order to improve the performance of our model, we created the following new features to provide more information to our model :

- Mean and standard deviation of other weather variables
- Cumulative precipitation of neighboring stations per day and their averages : This provides spatial information around a station
- Cumulative precipitation for previous days and the average of these values : This provides temporal information about a station

2.3 Missing data

During the processing of the data, we found that large parts of the ground station observation data were missing, as shown on Figure 1. To solve this problem we considered two possible courses of action : either we could use imputation methods to fill in the missing values, or we could use a model that can tolerate missing values, e.g. XGBoost. We opted to first go for the latter method since this would allow us to test a model quickly. At a later stage, we could always impute the missing values in order to compare with other models.

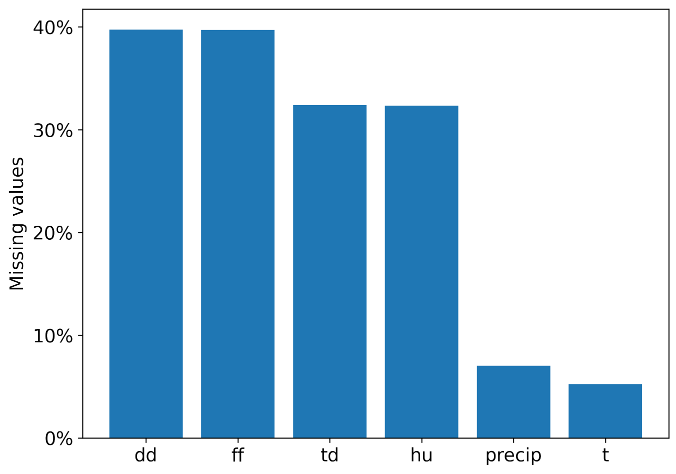

FIGURE 1 – Bar chart of missing data for variables in file X_station_train

3 Choice of models

3.1 Extreme Gradient Boosting(XGBoost)

Our first idea was to use the XGBoost algorithm for regression. In essence, the model consists of a particular set of small decision trees and combines these trees to predict the daily rainfall. This seemed like a natural choice for a few reasons. First off, XGBoost has historically had high performances in other Kaggle competitions, becoming one of the more popular algorithms for data scientists. Secondly, XGBoost performs especially well in the case of structured data such as ours. We also chose this model specifically because it handles missing data efficiently compared to deep-learning algorithms. As mentioned, we noticed a missing value percentage of close to 40% for some variables.

In terms of implementation, we used the XGBRegressor from the `xgboost` package for Python.

Before evaluating the performance of our model, we need to define which loss function is best suited for training.

3.1.1 Loss function

As stated on the competition page, the performance of our predictions would be be measured with the Mean Absolute Percentage Error (MAPE) metric, which is defined as :

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}|\frac{y_i - \hat{y}_i}{y_i}| \tag{1}$$

where $\hat{y}_i$ is the predicted value and y_i the target value. This affected our choice of loss function. The most frequently used loss for regression problems is Mean Squared Error (MSE). But since the performance of our model would be evaluated with a percentage error, we believe that the Mean Squared Logarithmic Error (MSLE) is a better suited loss function. It is defined as :

$$\text{MSLE} = \frac{1}{n}\sum_{i=1}^{n}(log(y_i + 1) - log(\hat{y}_i + 1))^2 \tag{2}$$

The advantage of the MSLE loss compared to MSE loss is that by applying the log function to the target and prediction we are more directly minimizing the relative error compared to the target y_i. This can be seen with the following rough calculation. Suppose that the prediction takes the form of $\hat{y}_i = \epsilon_i y_i$ where ϵ_i denotes the error. Thus,

$$\begin{aligned}
\text{MSLE} &= \frac{1}{n}\sum_{i=1}^{n}(\log(y_i + 1) - \log(\epsilon_i y_i + 1))^2 \\
&\approx \frac{1}{n}\sum_{i=1}^{n}(\log(y_i + 1) - \log(y_i + 1) + \log(\epsilon_i))^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}\log(\epsilon_i)^2
\end{aligned}$$

In this way, we can see that minimizing the MSLE is approximately equivalent to minimizing the relative error of the prediction.

Another approach is to directly use the MAPE metric as the loss function. We also tried this approach, but to use a custom loss function with XGBoost, we need to supply both the gradient and the hessian of the function. The problem is that the hessian of the MAPE function is zero, and when we tried to train the model with this loss function, XGBoost could not use it and the error stayed constant. We also tried to "trick" the XGBoost algorithm to use the simpler gradient descent method instead by supplying a vector of ones as the hessian. This allowed XGboost to update the model, but the convergence was very slow and the performance was nowhere near the one we achieved using MSLE loss. A possible compromise could maybe be to use a combination of MSLE and MAPE as the loss, in which case the loss function would have the hessian of the MSLE while still penalizing the MAPE. We did not have time to test this combined loss function, but it may be considered for future development. For this reason, even if the MSLE does not directly minimize the MAPE, we still believe it is better suited as the loss function compared to the MAPE function alone. A situation where we did use the MAPE function was during hyperparameter tuning, where we used it as the score to select the best performing model.

3.1.2 Bayesian optimization

There are several hyperparameters of the XGBoost algorithm that can be tuned. We will tune the following parameters in an attempt to improve the performance of our model :

- n_estimators
- max_depth
- learning_rate
- colsample_bylevel
- gamma
- min_child_weight
- max_delta_step
- colsample_bynode

With this large number of parameters, there are many possible combinations to consider when tuning. This is why we used Bayesian optimisation with the help of the BayesSearchCV algorithm in the `skopt` package in Python. In particular, we chose the MLSE as the loss function for the XGBoost algorithm, and the MAPE as the scorer for the BayesSearchCV algorithm. Each model was evaluated using three-fold cross-validation. The parameter space we considered is available in the parameter_space.json-file and the resulting optimization is presented on Figure 2. We found *max_ depth* to be the hyperparameter with the largest marginal effect on the model performance.

The optimal point found by the algorithm corresponds to a *max_ depth* of 10. This corresponds to the maximum depth allowed in our parameter space, and suggests that increasing the maximum depth any further could also improve our performance. However, we began experiencing problems with the GPU memory when testing deeper trees, so we decided not to allow for deeper decision trees in the XGBoost algorithm.

Thanks to Bayesian optimisation, we found an optimal combination of hyperparameters on the available training dataset, resulting in a promising MAPE score of 27.5 on our test data. When submitting the model to evaluate on the Kaggle-dataset, we did not see any improvement from our previous record of 30.54. Despite attempts to reduce the complexity of our model, we did not achieve as good results as we hoped after tuning parameters.

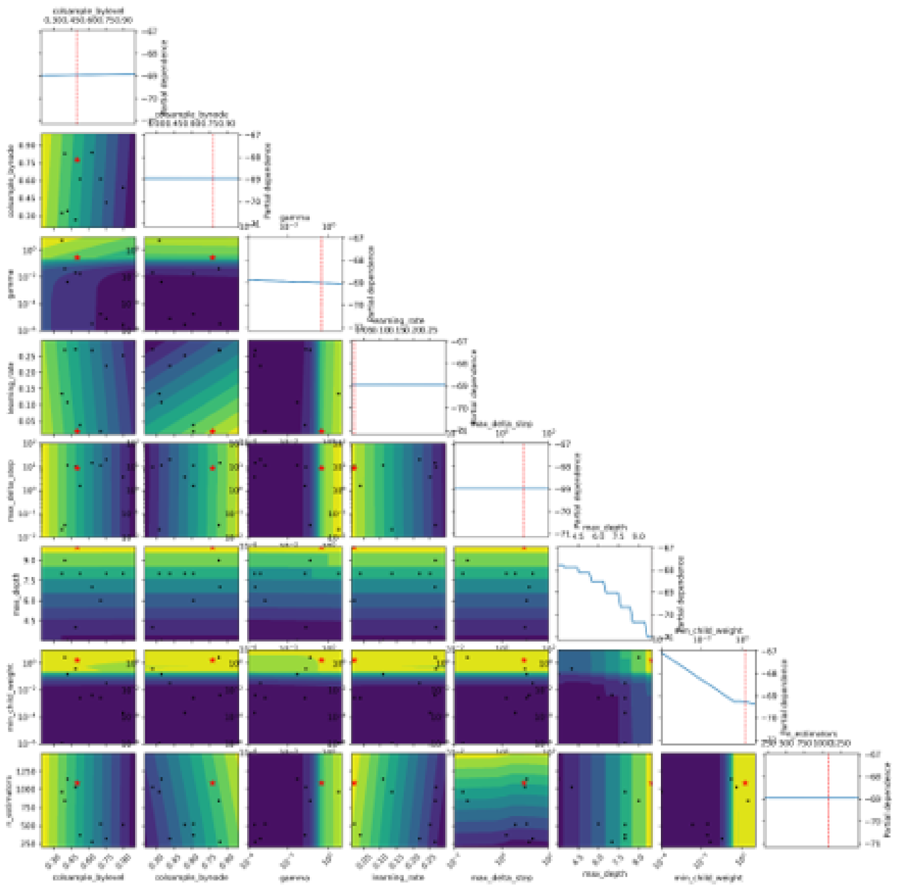

FIGURE 2 – Bayesian optimisation for XGBoost

3.2 Convolutional Neural Network

As an alternative to XGBoost, we have considered a Convolutional Neural Network (CNN). This is better suited to our problem than regular MLPs since we have both temporal information in terms of hourly weather measurements and spatial information from the geographical position of the weather stations.

Before training a CNN model, we needed to address the problem of missing data. We used the fillna method of the pandas Dataframe grouped by the station id, specifying the method argument first as "ffill" and then "bfill". This fills in the missing values for each station with the last observed value for that station, and if there are any missing values left, they are filled in with the first observed value for that station.

3.2.1 First version of CNN

For our first implementation of a CNN, we used the Conv1D layer from `keras` with input dimension (24,6) corresponding to 24 hourly measurements of the 6 weather variables. We joined two Conv1D layers with 32 filters and kernel size of 5 to a fully connected network predicting the rainfall of the following day. This model performed decently on our test data, with a MAPE score of 30.45.

To improve our model, we wanted to implement a 2D Convolution block which could be applied on the spatial data of the AROME model.

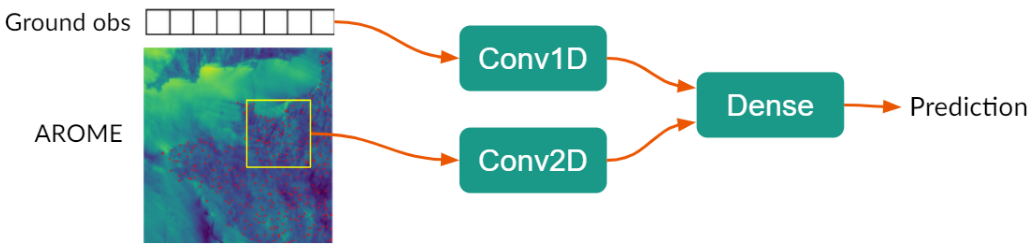

FIGURE 3 – A diagram of the planned architecture of our CNN model

3.2.2 Second version of CNN

Although we did not have the time to implement the second version of our CNN model, we want to describe our plans for the implementation of the model in this section. The second version of the model builds upon the first version, adding a 2D convolution part using the predictions of the AROME model. Our plans for the architecture of this model are shown in Figure 3. The reason we considered a CNN in the first place was to exploit the spatial data provided by the AROME and ARPEGE models, but we found it difficult and time-consuming to manipulate these files. To generate the data we would use to train the model, our plan was : for each station and each day, we take a slice with a fixed size (e.g. 32x32) of the AROME data for that day centered at the given station. This would be the input of the 2D convolution part. A problem with this method is that for the stations at the edges of the domain of the AROME predictions, some of the values in the slice window will be missing. A possible way to solve this would be to fill in the values, for example with the nearest value or by mirroring the values. The generation of this dataset would take a very long time, since the data from the AROME and ARPEGE is quite large (40GB !)

4 Conclusion

In the course of this project, we tested two very different predictive models : an XGBoost regressor and a convolutional neural network. To train these models and improve our results, we had to make use of techniques seen in courses of AI Frameworks and HDDL, but we also used some new techniques such as Bayesian optimisation. The most time-consuming part of the challenge was the processing and preparation of the data, which required a lot of trial and error. We found that feature engineering and efficient use of the data were just as important for the performance of the prediction as the tuning of the parameters.

As we had expected when we first approached this problem, the XGBoost model gave us the best performance with a final 30.44 MAPE score. The other benefit of the XGBoost model was that we did not have to impute the missing values, which were numerous. The main problem we encountered with the XGBoost model was that when we tried to tune the parameters of the model using Bayesian optimisation, it resulted in overfitted models, even when using early stopping. Our convolutional neural network did not achieve as good performance as the XGBoost model, but we believe that by implementing our plans for the 2D convolution part of the network, the performance of the CNN model could surpass that of the XGBoost model.