INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Machine Learning

Report

Students

Minh Hai NGUYEN, Cam Thanh Ha LE, Ayoub EL ARROUD EL HADARI and Paul CORBALAN
4 GMM A

Supervisor : Béatrice LAURENT

May 20, 2022

Contents

Introduction

We have a dataset consisting of 688 meteorological observations at a given station. The aim of the project is to predict the amount of rainfall over the next day, based on the following observed meteorological parameters during the current day:

- date: the date of the current day
- ff: the wind speed (in $m.s^{-1}$)
- t: the temperature (in *Kelvin* K)
- td: the dew point (in K). The dew point is the temperature the air needs to be cooled to (at constant pressure) in order to achieve a relative humidity (RH) of 100%. At this point the air cannot hold more water in the gas form.
- hu: the humidity (in %)
- dd: the wind direction (in degrees);
- precip: the total amount of precipitation (in $kg.m^{-2}$)

As well as meteorological parameters for the next day forecasted by the *MétéoFrance* **Arome Model** :

- ws_arome: the wind speed (in $m.s^{-1}$)
- p3031_arome: the wind direction (in degrees)
- u10_arome and vu10_arome: the U (from West to East) and V (from South to North) wind components (in $m.s^{-1}$) at the vertical level of $10m$;
- t2m_arome: the temperature at the vertical level of $2m$ (in K)
- d2m_arome: the dew point at the vertical level of $2m$ (in K)
- r_arome: the humidity (in %)
- tp_arome: the total amount of precipitation in ($kg.m^{-2}$)
- msl_arome: the sea level pressure (in Pa)

The response variables are :

- rain (quantitative): the total amount of rainfall over the next day (in $kg.m^{-2}$)
- rain_class (qualitative): an artificially created categorical variable with three classes that are : no_rain (if rain $= 0$), low_rain (if $0 <$ rain ≤ 2), and high_rain (if rain > 2)

In other words, **our objective is to solve the following classification problem : predict the rain quantity (rain_class) during the next day from the previous explanatory variables**

1 Data analysis

Before going any further, it is important to have a good understanding of the data set we are working on.

1.1 Uni-dimensional Analysis & Data Transformation

The distribution of all quantitative variables is presented in the Fig. 1.

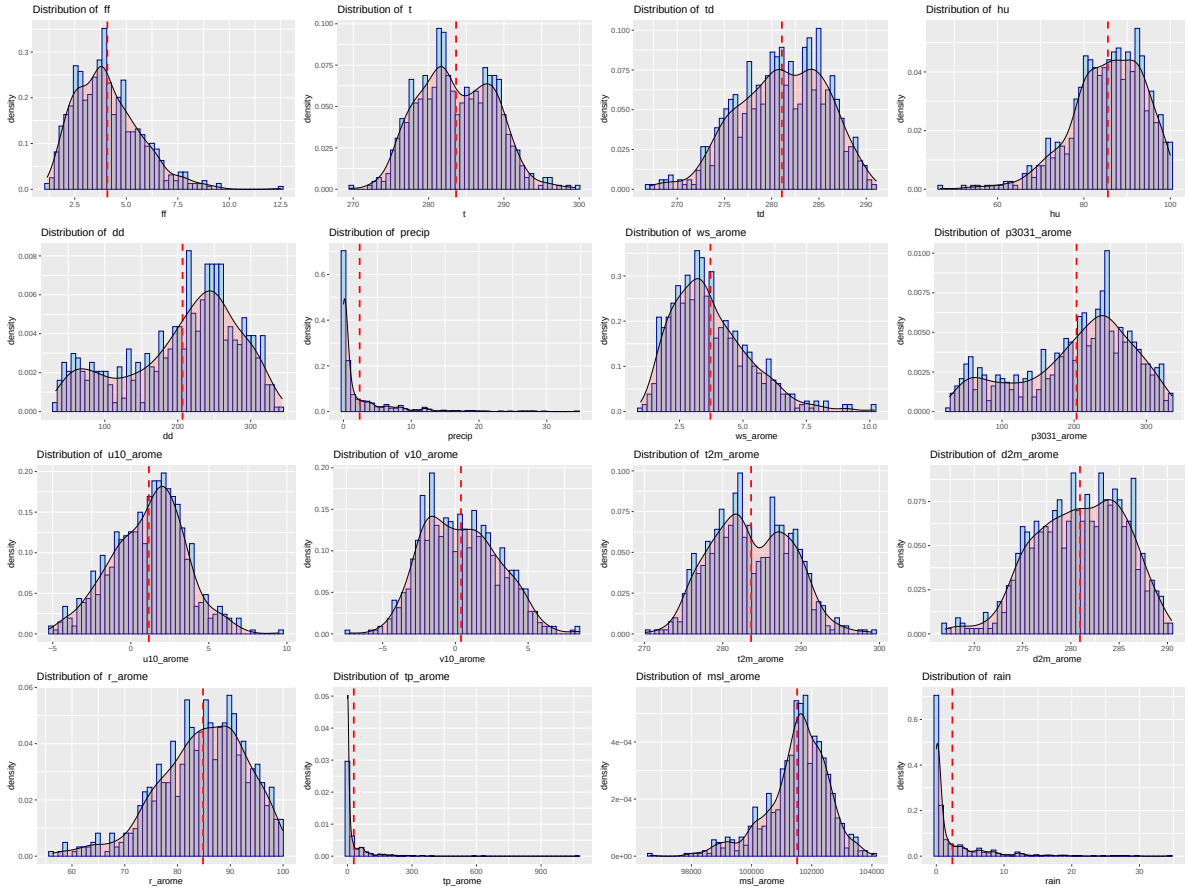


Figure 1: Histograms of Quantitative Variables

We see that variables are measured in different scales, and some of them are not centered or symmetric. We decided therefore to center and normalize the data. More details are provided about this process in section 2.1.

Moreover, we know that when it comes to rainfall, the useful information to grasp from the `date` variable is the month (see the distributions of rain by month in Fig.3). Therefore, there is no need to conserve the `dd/mm/yyyy` format: we can simply replace it by a new categorical variable called `month`. Moreover, when we visualize histograms of the different quantitative variables, we notice that the three variables `precip`, `tp_arome` and `rain` have strongly positively skewed distributions . In order to fix this issue, we are going to re-express those variables using `logarithm`, or more precisely $\log(. + 1)$ since the variables can take the value 0. We have named this variable `rain_log`. The comparison of the distribution of `rain` and `rain_log` is given in the Fig. 2, we also see that this simple log transformation reduce dramatically the number of outliers.

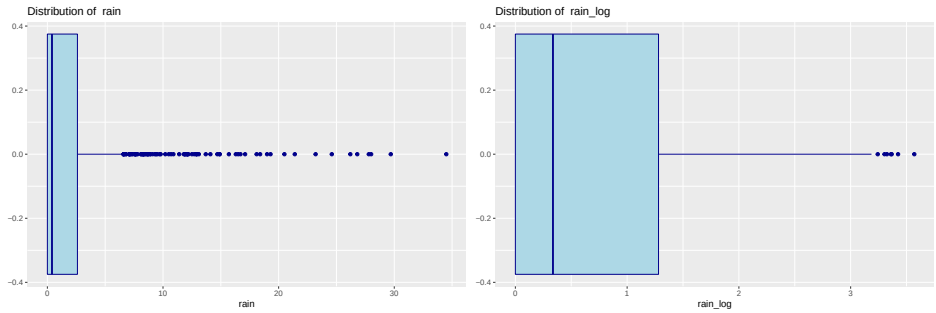


Figure 2: Boxplots of `rain` and `rain_log`

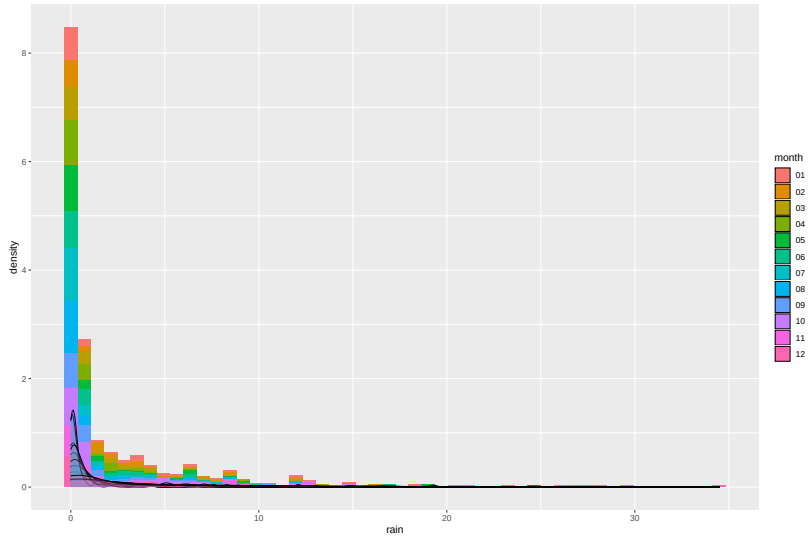


Figure 3: Histograms of rain variable by months

On the other hand, with regard to the distribution of rainfall by month, we note that the least rainy months are August to October and the rainiest months are January to March.

1.2 Multi-dimensional Analysis

The correlation matrix (see Fig.4) shows that some variables are highly correlated with others. For example, variables t and td are both correlated with variables $t2m\_arome$ and $d2m\_arome$, as well as with one another. As a matter of fact, the last two variables are also correlated with one another. Same applies to ff and ws_arome, which makes sens since wind speed in the current day affects wind speed the next day.
The variable that seems to have the most influence on our response variable $rain$ is msl_arome, with a correlation coefficient of -0.4, meaning that it's more likely to rain when the pressure on sea level decreases.

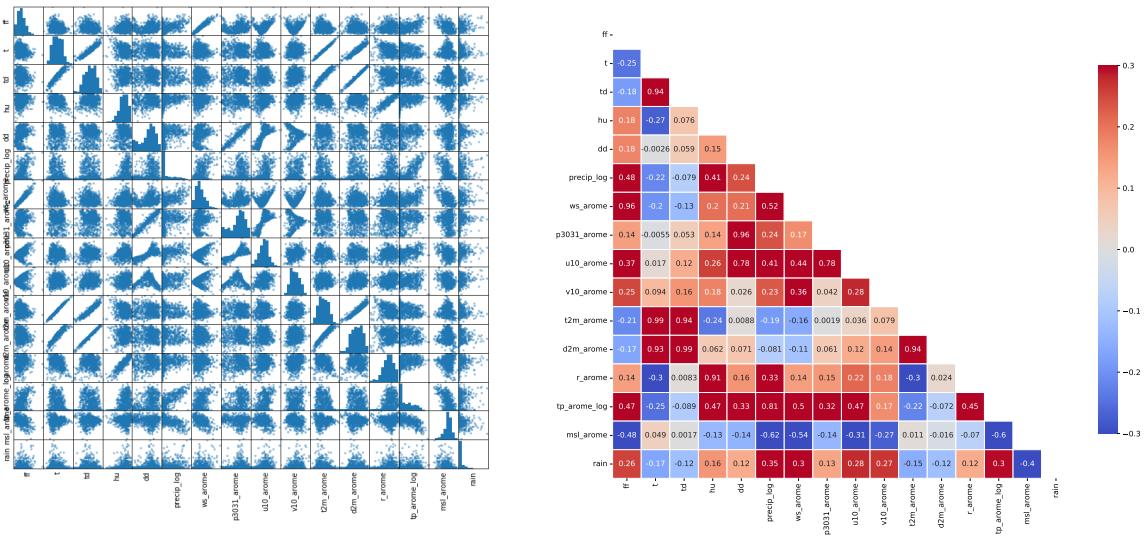


Figure 4: Linear correlation between variables

But unfortunately, the response variable `rain` is not linear correlated with other variables.

1.3 Principal Component Analysis (PCA)

1.3.1 Choice of PCA variables

We now proceed to Principal Component Analysis (PCA). In order to achieve a 95% cumulative explained variance, we need to consider 6 principal components out of 15.

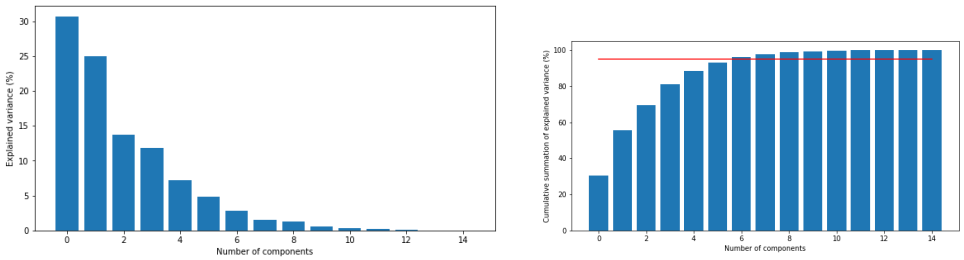


Figure 5: Barplot (left) and cumulative summation (right) of explained variance at every component

1.3.2 Presence of clusters

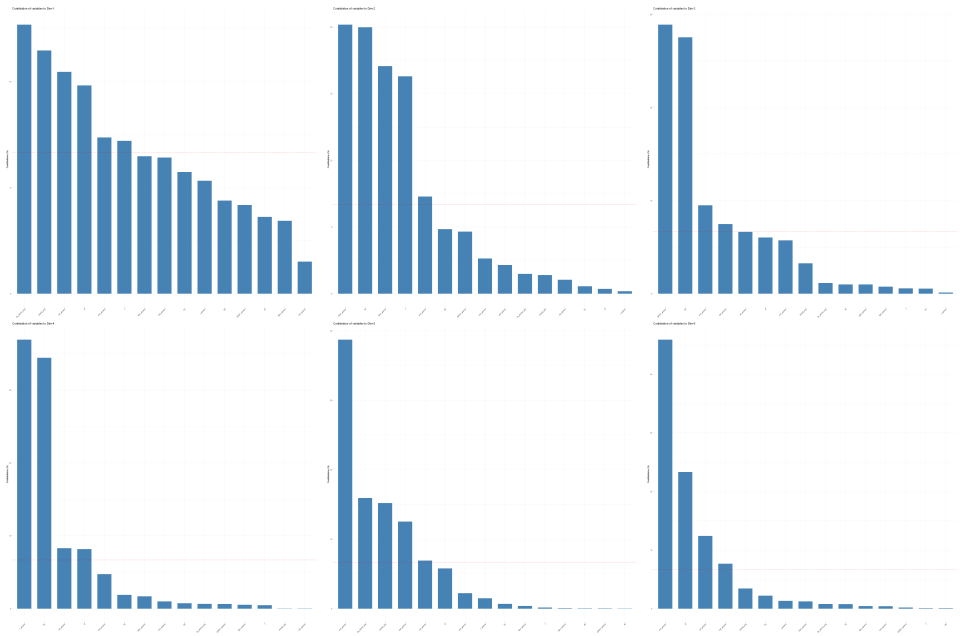


Figure 6: Correlation of the first six PCA components with each components

To analyse the presence of clusters, we will observe the composition of the PCA components:

1. This one is essentially composed of `tp_arome_log`, `precip_arome`, `ws_arome`, `ff`, `u10_arome`. and `t`.
   - `tp_arome_log`, `precip_arome`: corresponds to the precipitation.
   - `ws_arome`, `ff`, `u10_arome`: wind speed measures.
2. This one is essentially composed of `tp_arome_log`, `precip_arome`, `ws_arome`, `ff`, `u10_arome`. In regard of those variables, this is a component that represents temperature.
   - `d2m_arome`, `td`: temperature dew point information.
   - `t2m_arome`, `t`: temperature information.
3. This one is essentially composed of `p3031_arome`, `dd`, `u10_arome`. Represents the overall wind direction.
4. This one is essentially composed of `p3031_arome`, `hu`, and in a minor way `ws_arome`, `tt`. Mostly this component represents the humidity and to a minor extent the wind speed.
   - `p3031_arome`, `hu`: humidity in air.
   - `ws_arome`, `tt`: wind speed measures.
5. This one is essentially composed of `v10_arome`, and in a minor way `tp_arome_log`, `precip_log`, `msl_arome`. It corresponds mainly to wind speed measures from South to North and precipitation to a lesser extent.
   - `v10_arome`: wind speed measures from South to North.
   - `tp_arome_log`, `precip_log`: corresponds to the precipitation.
   - `msl_arome`: the sea level pressure.
6. This one is essentially composed of `v10_arome`, `ff`, `ws_arome`, and in a minor way `msl_arome`. It represents the overall wind speed.
   - `v10_arome`, `ff`, `ws_arome`: wind speed.
   - `msl_arome`: sea level pressure.

1.3.3 Link with the `rain_classes`

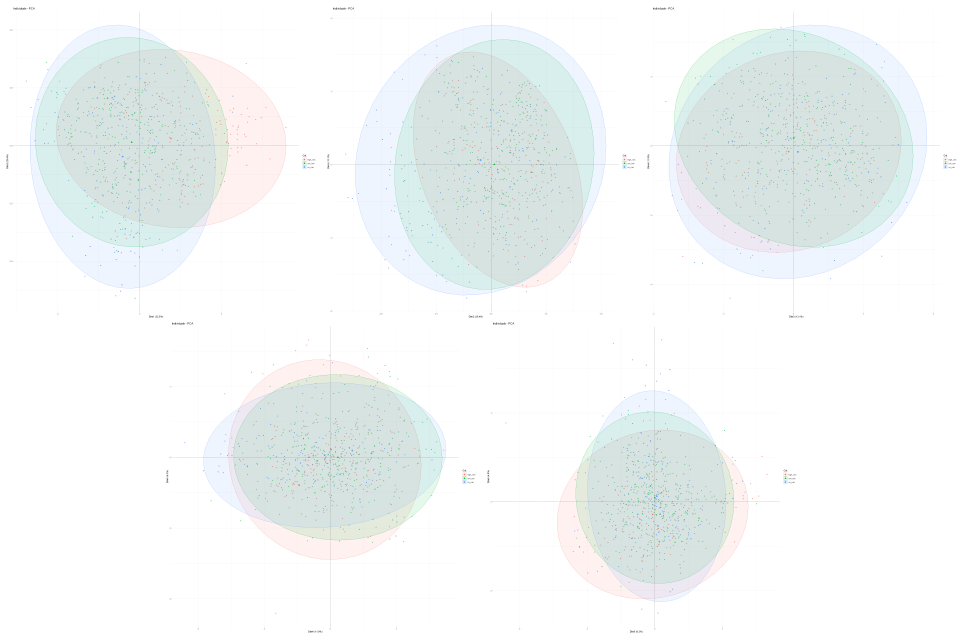


Figure 7: Maps of individuals of the first six PCA

The maps of individuals above do not allow us to detect real clusters with `rain_classes` at the level of individuals.

2 Preparation for Machine Learning Models

2.1 Data Preprocessing and Data Splitting

Since the explanatory variables are measured at different scales and have different ranges, we need to apply normalisation (except for `month`) to prevent creating a bias. Moreover, we will use some models with penalisation in order to ensure that **the penalty treats features equally**. Furthermore, in order to perform and test the different classification methods, we decide to split our data into two samples:

- `train_set` : a training set consisting of 80% of the initial data, obtained randomly.
- `test_set` : a testing set that contains the remaining 20% of the initial data.

This split is essential to compare and choose the most suitable model for our data. The learning data (or training set) will allow us to build the model and the test data will allow us to evaluate the capacity of prediction (or capacity of generalization) of the built model.

We also save this split as a `train_set.txt` and `test_set.txt` to assure the reproducibility of the results and for comparing the implementations in R and Python.

2.2 Reproducibility

We can remark that many Machine Learning algorithms have a random component. Therefore, during this project we set the `random_seed` to be 42 (just a famous number on the Internet) on Python as well as on R.

An another remark is that the results from R and Python might be different even if the data is the same. This is due to the randomness and the problem of numerical stability in the implementations. Nevertheless, this difference is generally negligible.

3 Classification problem

In this section, we consider the classification problem, we aim to use the explanatory variables to classifier the `rain` into 3 classes : `no_rain`, `low_rain` and `high_rain`.

Before diving into the details, we notice here that we removed the `rain` (and also `rain_log`) from the data during this section and kept all other variables, including the dummy variables for the `month` feature. We want to keep this effect of month because we have seen in the Data Analysis section that the distribution of `rain` is different in different months, and also in real life, rainfall changes during the year so the month must have a certain effect.

To evaluate the performance of our models, we will use different metrics for this classification context:

- The **accuracy**, which represents the percentage of values correctly predicted by the model.
- The **recall**, which represents the percentage of well positive predicted values on total positive values.
- The **precision**, which represents the percentage of well positive predicted values on total positive predicted values.
- The F_1**-score**, which gives the combined result of Precision and Recall using a Harmonic mean.

3.1 K-nearest neighbors

Description : K-nearest neighbors is a non-parametric supervised learning method which can be used in our classification context. The mechanic of this algorithm is based on a distance measure and for a new individual, its label is predicted using a majority vote from the labels of its K-nearest neighbors associated with a distance. In our problem, the dataset is quite small so this is a good idea to start with.

We used the usual Euclidean distance as the distance measure and we use cross-validation to determine the number of neighbors. The results (using different metrics) are given in the following table, for the optimal number of neighbors which is **18**.

Prediction	Observation			Recall	Precision
	high_rain	low_rain	no_rain		
high_rain	20	11	11	0.606	0.476
low_rain	12	37	23	0.638	0.514
no_rain	1	10	13	0.277	0.542

- The prediction accuracy: 0.507
- The recall-score: 0.507
- The F_1-score: 0.491

Commentary : The results are around 50 percents for the prediction accuracy on the test set, which is not very bad for a such simple first approach.

3.2 Decision tree

Description : In this section, we are going to use the CART method. This method consists of separating our data into two classes by choosing a variable and a threshold that will minimize the heterogeneity of the two classes obtained. By repeating this separation process we obtain what we call a tree. The goal here is to create a model that predicts the class of a target variable (rain) by learning simple decision rules inferred from the data features. A tree can be seen as a *piecewise constant approximation.*

Since the decision tree heavily depends on how we split a node and on the structure of the tree (the depth of the tree, number of leaves, ...), we used cross-validation to find the optimal value for each hyper-parameter of our model.

We investigated the effect of different hyper-parameters and tried to find their optimal value using cross-validation, such as:

- `max_depth` : The maximum depth of the tree, by default, then nodes are expanded until all leaves are pure.
- `min_samples_split`: The minimum number of samples required to split an internal node.
- `min_impurity_decrease`: A node will be split if this split induces a decrease of the impurity greater than or equal to this value.

Using the cross-validation, the optimal tree is found and the performance on the test set is reported below, along with the tree:

Prediction	Observation			Recall	Precision
	high_rain	low_rain	no_rain		
high_rain	14	6	4	0.424	0.583
low_rain	17	39	29	0.672	0.459
no_rain	2	13	14	0.298	0.483

- The prediction accuracy: 0.486
- The recall-score: 0.472
- The F_1-score: 0.486

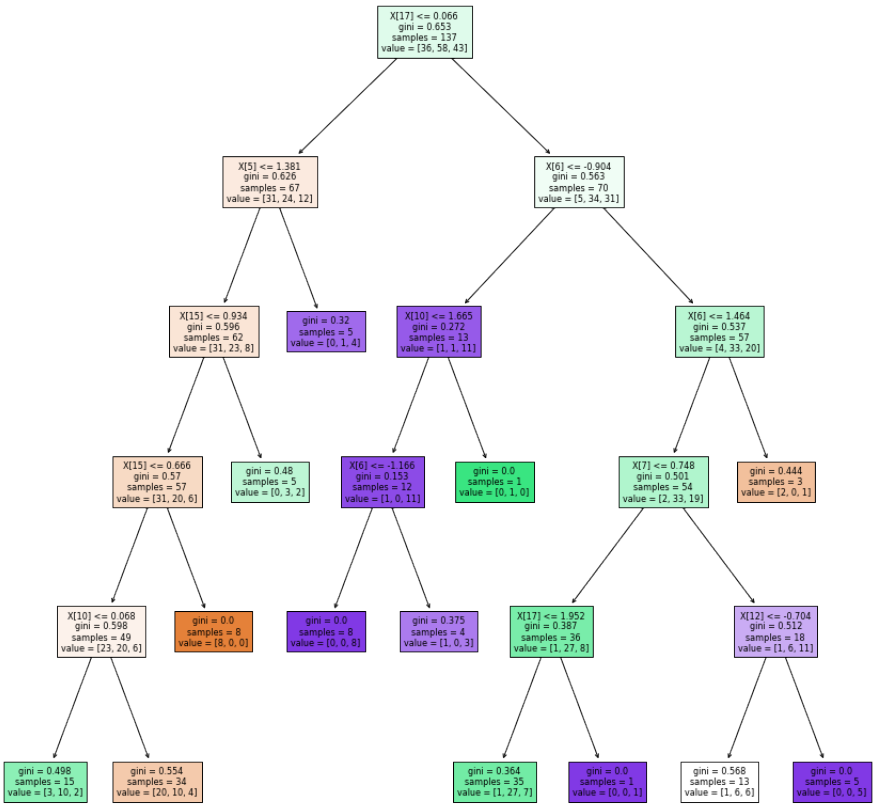


Figure 8: Trained Decision Tree

3.3 Random Forest

Description: Random forest is a supervised machine learning algorithm which is a boosting version of decision trees. It creates decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting.

Many hyper-parameters are tuned using cross-validation in this section:

- `n_estimaters`: number of decision trees in the forests, which also defines the complexity of the model, usually a large number. We had tested for 200, 300, 400 and 500 trees and the best result given by cross-validation was 300 trees.
- Other hyper-parameters presented in the section 3.2 have been also tested using cross-validation.

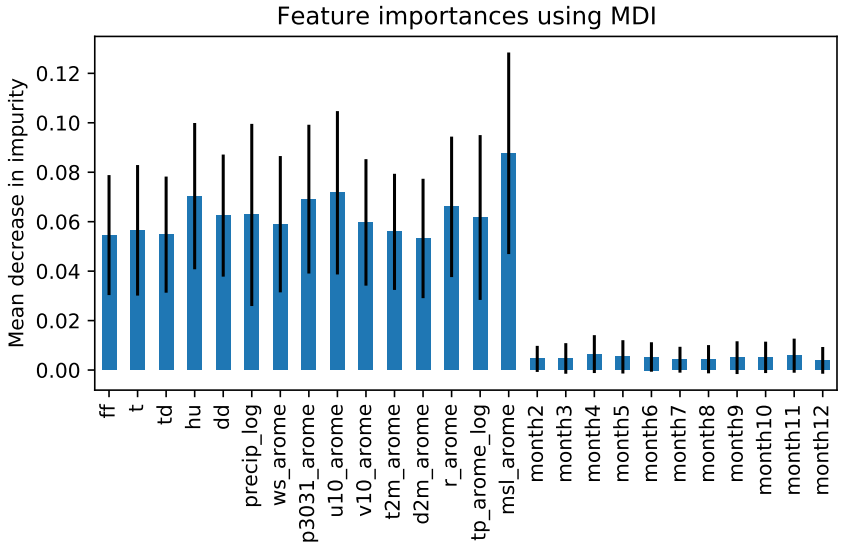


Figure 9: Feature Importance

We can see that the month variables play a miner role. We remark that variables with most influence on the prediction are `u10_arome`, `hu` and `msl_arome`, respectively the U wind component at 10m, humidity, and sea level pressure; meanwhile other variables are at the same important level. The result is given below:

Prediction	Observation			Recall	Precision
	high_rain	low_rain	no_rain		
high_rain	20	12	1	0.61	0.48
low_rain	12	35	11	0.60	0.49
no_rain	10	24	13	0.28	0.52

- The prediction accuracy: 0.49
- The precision: 0.50
- The recall-score: 0.50
- The F_1-score: 0.48

Commentary: By using boosting with many decision trees, we got a slightly better result comparing to the decision tree, but the difference is not significant.

3.4 Support Vector Machine (SVM)

SVM constructs hyperplans (linear) to separate some groups of data. Thanks to the kernel trick, SVM can establish non-linear boundaries and therefore can perform non-linear classification. In this part, we will perform a Support Vector Classifier with different kernels (linear, polynomial, radial or sigmoid). The kernel trick works thanks to the Reproducing Kernel Hilbert Space (RKHS).

The Support Vector Machine can be reformulated as the following optimisation problem:

$$\text{minimizing } \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \text{ w.r.t } (w, b, \xi) \text{ under constraints } \xi_i \geqslant 0 \text{ and } y_i(\langle w, x_i\rangle + b) \geqslant 1 - \xi_i$$

Intuitively, we're trying to maximize the margin (by minimizing $\|w\|$) while incurring a penalty when a sample is misclassified or within the margin boundary. Ideally, the value $y_i(\langle w, x_i\rangle + b)$ would be $\geqslant 1$ for all samples, which indicates a perfect prediction. But problems are rarely perfectly separable with a hyperplane, so we allow some samples to be at a distance from their correct margin boundary. C is a tuning parameter of the SVM algorithm which controls the strength of this penalty. If C is small, the number of miss classified points increases, and reversely.

By moving to the dual problem and using the results from RKHS, we can replace the scalar product by the value of kernel, i.e, we replace $\langle w, x_i\rangle$ by $k(w, x_i)$ where k is a kernel. Therefore, we can use SVM as a non-linear model using other kernels.

3.4.1 Linear Support Vector Machine

Description: In this case, we use directly the scalar product, so the kernel trick is not applied here. The optimisation problem 3.4 is reformulated using the *hinge loss* for our classification problem.

We used cross-validation to tune the C hyper-parameters over the grid and the optimal C found is 0.06. The performance on the test set is given below

Prediction	Observation			Recall	Precision
	high_rain	low_rain	no_rain		
high_rain	25	7	1	0.758	0.556
low_rain	11	36	11	0.621	0.571
no_rain	9	20	18	0.383	0.600

- The prediction accuracy: 0.572
- The recall-score: 0.563
- The F_1-score: 0.572

Commentary: The performance is quite good. We got 0.572 for the prediction accuracy and other metrics are at the same level. But the recall and F_1-score for the `no_rain` class is significantly smaller than the two other classes. Which means that our algorithm did not well predict the `no_rain` class. There are particularly many points predicted to be `no_rain` whereas they're actually at the `low_rain` class.

3.4.2 Support Vector Machine with polynomial kernels

Description: The kernel trick is used here with the polynomial kernel

$$k(x, x') = (\gamma\langle x, x'\rangle + r)^d$$

In this part, we used cross-validation to find the optimal value of hyper-parameters γ, r and d, and searched for d in $\{2, 3\}$, i.e we find a polynomial kernel of degree 2 or 3. **Results on the testing set:**

Prediction	Observation			Recall	Precision
	high_rain	low_rain	no_rain		
high_rain	20	10	7	0.606	0.541
low_rain	13	38	26	0.655	0.494
no_rain	0	10	14	0.298	0.583

- The prediction accuracy: 0.522
- The recall-score: 0.510
- The F_1-score: 0.520

Commentary: We got almost the same results for polynomials of degree 2 and those of degree 3. But the results are not good enough compared to the linear case, even in the training sample.

3.4.3 Support Vector Machine with radial kernel

Description: The radial kernel is, for γ positive is a hyper-parameter to tune

$$k(x, x') = \exp\left(-\gamma \|x - x'\|^2\right)$$

The result on the testing set for the best model using cross-validation is given below:

Prediction	Observation			Recall	Precision
	high_rain	low_rain	no_rain		
high_rain	21	9	10	0.636	0.525
low_rain	11	40	26	0.690	0.519
no_rain	1	9	11	0.234	0.524

- The prediction accuracy: 0.522
- The recall-score: 0.497
- The F_1-score: 0.520

Commentary : We used cross-validation to tune carefully the hyper-parameter but even in the best case, we just got around 0.57 prediction accuracy on the training set and only 0.52 on the test set. Which is indeed smaller than the linear case.

3.4.4 Support Vector Machine with sigmoid kernel

Description: The sigmoid kernel is given by:

$$k(x, x') = \tanh\left(\gamma \langle x, x' \rangle + r\right)$$

Results on the testing set are given below:

Prediction	Observation			Recall	Precision
	high_rain	low_rain	no_rain		
high_rain	26	12	8	0.788	0.565
low_rain	7	42	35	0.724	0.500
no_rain	0	4	4	0.085	0.500

- The prediction accuracy: 0.522
- The recall-score: 0.456
- The F_1-score: 0.522

Commentary: Compared to previous classification results, the performance is generally at the same level. We realized that it's very hard to fit a model on this dataset, even in the training phase using SVM models.

3.4.5 Commentary on the SVMs

The best performance is obtained using the linear SVM Classifier. Still, the result is not extraordinary: we just got around 0.57 for the prediction accuracy on the test set. In fact, this means that the dataset is not linearly separable (in the usual Euclidean space). Thanks to RKHS, we mapped the data into other Hilbert spaces using some well-known kernels, but the result is even worse than the linear case. This could be explained by the fact that the data is not separable in these Hilbert spaces neither or is of poor quality. Either way, the training set does not have a good enough capability to generalize.

3.5 Neural Network

Description: In this section we will adapt some Neural Network architectures to our classification problem. We will use some Neural Networks with one-hidden layer or multiple hidden-layers (also called Multi-layer Perceptron classifier), using different activation functions. But all models will use the same `softmax` activation function for the output layer which gives us the probabilities of belonging to classes.

The learning process consists of minimizing a loss function using some new gradient-based optimisation algorithms (Adam, LBFGS) thanks to the back-propagation technique.

3.5.1 One Hidden-layer Neural Network

In this part, we try to fit a one hidden-layer neural network to our dataset. The *Universal Approximation Theorem* assures that any bounded and regular function can be approximated at any given precision by a neural network with one hidden layer containing a finite number of neurons, but this result does not help us in practice.

We use cross-validation to find the optimal number of neurons in the hidden-layers. The optimal value is 5 using ReLU activation. We got 0.65 for the prediction accuracy in the training set, the performance in the test set is reported below:

Prediction	Observation			Recall	Precision
	high_rain	low_rain	no_rain		
high_rain	25	7	1	0.76	0.56
low_rain	12	33	13	0.57	0.52
no_rain	8	24	15	0.32	0.52

- The prediction accuracy: 0.53
- The recall-score: 0.55
- The F_1-score: 0.53

Commentary: The performance on the training set seems to be promising, but in reality, the performance on the test set is not very good, just around 0.53, pretty similar to previous methods.

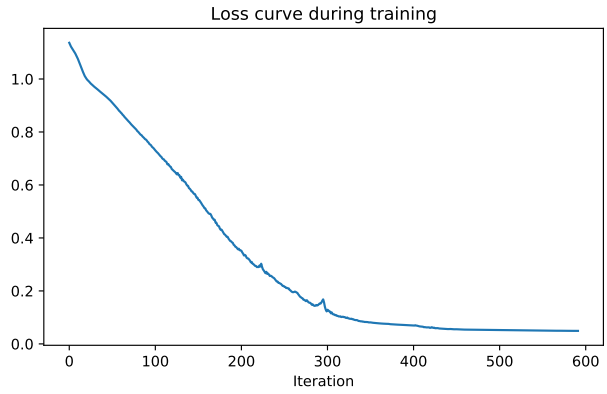
3.5.2 Multi-layer Perceptron Classifier

[1] Similarly to the previous section, the MLP just has more hidden-layers, which means a more sophisticated architecture and it could approximate more complicated function.

After testing for different architectures, here are some important remarks

[1] This section is only implemented in Python, not in R because the NNs are not very good supported in R

- If we just fit a sophisticated model with many parameters (weights and biases), i.e the model is over parameterized, we can get very high performance on the training set (even 100 percents). But unfortunately, we get a worse performance on the test set and the model generalizes badly. This is the **overfitting** phenomenon. For example, a Neural Network with 6 hidden-layers of sizes $20, 25, 15, 15, 10, 5$ respectively, using ReLU activation has a training accuracy equal to 1 but just has 0.46 on the testing set. The evolution of a such neural network is given as below



- **Early stopping** can prevent overfitting. We used early stopping during optimisation, we stop if the performance on validation set starts to decrease in some consecutive iterations, even when the loss function still decreases.
- The best model we found have a prediction accuracy on the test set equal to 0.57, which is as good as the best model for now, the linear SVM Classifier.

Prediction	Observation			Recall	Precision
	high_rain	low_rain	no_rain		
high_rain	25	7	1	0.76	0.56
low_rain	11	42	5	0.72	0.55
no_rain	9	27	11	0.23	0.65

  - The prediction accuracy: 0.57
  - The recall-score: 0.57
  - The F_1-score: 0.54

  And again, the performance on the `no_rain` is really worse comparing to two other classes.

3.6 Comparison of classification methods

In our case, the classification problem deals with a supervised Machine Learning problem. As mentioned previously, we evaluate the model goodness through some metrics such as Accuracy, Average Precision, Average Recall or F1 score. The results are reported in the Fig. 10.

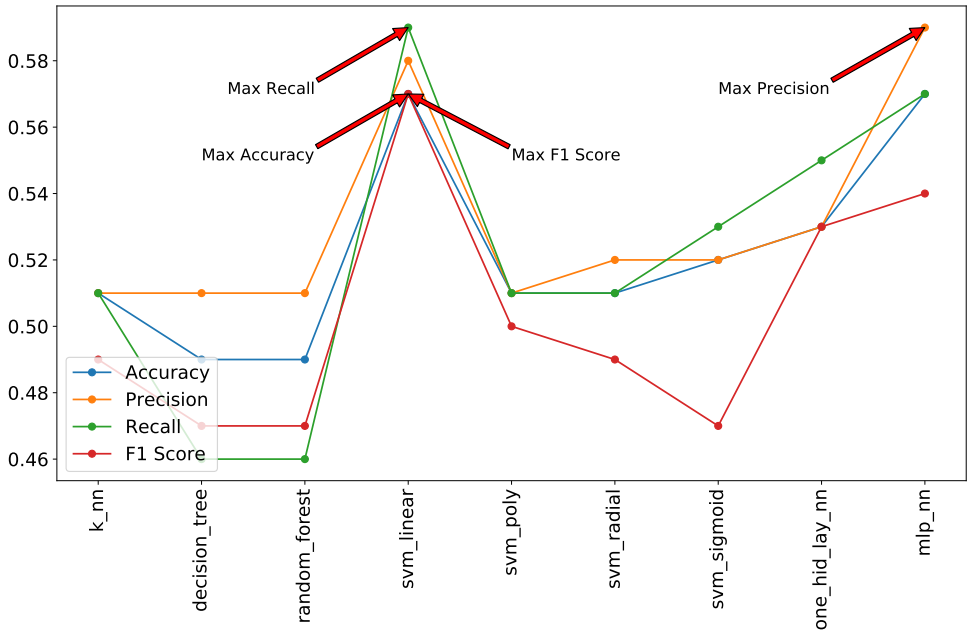


Figure 10: Classification performance

By comparing the obtained results of each method, we remark that the accuracy score is always lower than 0.6, which means the prediction is not really extraordinary, this can be explained by the fact that the data is not large enough to build a good model. Among all used methods, there is no significant difference: score values oscillate around 0.5.

We can notice that the SVM with linear kernel works better than SVM with other kernels, and even better than Random Forest, Decision Tree and One hidden-layer neural network. In fact, the SVM with non-linear kernels also work as good as linear SVM (even better) for detecting `high_rain` and `low_rain` classes, but they stack at distinguish the `no_rain` class, which is a common problem encountered with almost all models presented here. And this issue causes the average performance of these non-linear to be lower than the performance in the linear case.

The Multi-layer Neural Network also works better than other models, with a performance comparable to Linear SVM.

4 Regression and classification using the given thresholds

In this section, we focus on regression, but our final aim is of course classification using the thresholds. The response in this section is `rain` (and eventually `rain_log`).

Evaluation Metrics: To evaluate the performance of different methods, many metrics are used in this regression context

- Mean absolute percentage error (MAPE), defined by :

$$\frac{100}{n}\sum_{i=1}^{n}\frac{|Y_i - \hat{Y}_i|}{|Y_i|}$$

  The idea of this metric is to be sensitive to relative errors. Furthermore, the error can be arbitrarily high when `y_true` is small or when $\_true - y\_pred|$ is large.

- Mean Square Error: a risk metric corresponding to the expected value of the squared (quadratic) error or loss.

- R^2 **score** measures the goodness of fit of a model.
- Mean Absolute Error: a risk metric corresponding to the expected value of the absolute error loss or ℓ_1-norm loss.

Since in our data, the response `rain` could be zero, and as a matter of fact, this is the case of **208** individuals, therefore theoretically the MAPE is infinite. But the implementation in Python (with `scikit-learn`) can handle this using a numerical stability trick, more details can be found at Scikit-Learn User Guide. We implemented our own function for MAPE using the same trick, where we add an `epsilon` constant for the numerical stability (preventing division by zero).

We saw previously that there are many outliers in our data. So the Mean Absolute Error is more robust than MSE for dealing with outliers. Nevertheless, we used all of these metrics to measure the performance of models.

Using Regression for Classification:

Here in this regression setting, the outputs are numerical values. We will use these values to transform to the `rain_class` for the classification problem: We set the output to be `no_rain` if the `rain = 0`, `high_rain` if the `rain > 2`, and `low_rain` otherwise. In the regression setting, the response is a continuous variable so we allowed here some perturbations ϵ adding to the given thresholds. Based on this idea, we wrote a function transforming `rain` to `rain_class` to simplify the evaluation of classification problem. More details can be found in the notebook. Moreover, we used the same metrics to evaluate the performance in classification context.

4.1 Linear regression

This is the simplest model that we have to try in regression context. In this part, we tried to apply this method for two kinds of response variables: `rain` and `rain_log`

4.1.1 Linear regression without variable selection, applying to the `rain_log` response

To well evaluate and compare, noting that the variable response is in log scale, we should convert into the original scale

Testing error for classification

- MSE: 18.36
- MAE: 2.06
- MAPE: 1494003623307454.5
- R^2: 0.29
- Accuracy : 0.51

Prediction	Observation			Recall	Precision
	high_rain	low_rain	no_rain		
high_rain	23	11	7	0.70	0.56
low_rain	8	45	38	0.78	0.49
no_rain	2	2	2	0.04	0.33

Interpretation: The value of accuracy is low, the test errors are quite high, so the model is not well.

4.1.2 Linear regression without variable selection, applying to the *rain* response

Testing error for classification

- MSE: 21.31
- MAPE: 3012463523105789.0
- MAE: 2.71
- R^2: 0.18
- Accuracy : 0.44

Prediction	Observation			Recall	Precision
	high_rain	low_rain	no_rain		
high_rain	29	28	21	0.88	0.37
low_rain	2	24	18	0.41	0.55
no_rain	2	6	8	0.17	0.50

Interpretation: Comparing with `rain-log` response, we remark that the MSE error is higher and the accuracy is reduced. In general, linear model with both kinds of variables responses does not give us a good result. Our results are therefore consistent with the preliminary data analysis, specially the non-linearity property.

4.2 Variables selection

In this section, we will use some criterion such as BIC, Cp-Mallow and R^2 score to determine the variables having the most influence to use for fitting linear model

4.2.1 Variables selection for the *rain_log* response

First, let's see how many variables we can retain with each criterion. We retain 10, 7 and 2 variables for R^2, Cp and BIC criterion respectively. Next, we can see in more details what variables are kept exactly.

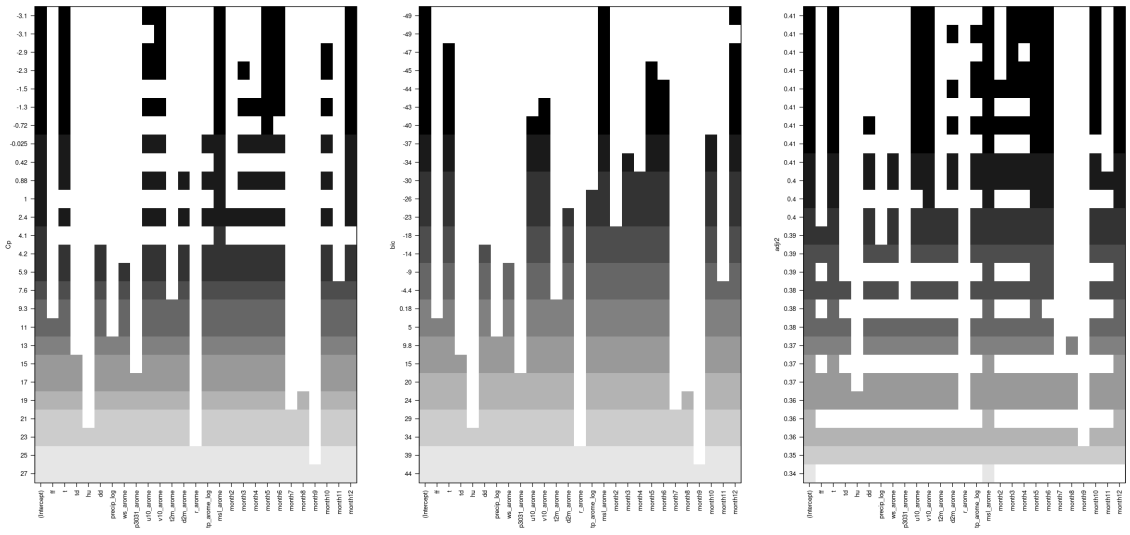


Figure 11: Variables selection by Cp, BIC and R^2 respectively

Through these graphics, by considering all 3 criterion, we decide to retain 6 variables : `t, u10_arome, v10_arome, tp_arome_log, msl_arome, month12`

Now, we fit a linear model using these 6 variables retained.

Training error for classification

- MSE: 16.732
- Accuracy: 0.511
- MAPE: 2326213.558
- MAE: 2.029
- R^2: 0.153

Prediction	Observation		
	high_rain	low_rain	no_rain
high_rain	70	26	7
low_rain	85	200	143
no_rain	0	8	11

Testing error for classification

- MSE: 18.313
- MAE: 1.971
- MAPE: 2972241.702
- R^2: 0.295
- Accuracy: 0.507

Prediction	Observation		
	high_rain	low_rain	no_rain
high_rain	23	11	4
low_rain	10	47	43
no_rain	0	0	0

Interpretation: Compared to the linear model without variables selection, the current model does not seem better. There is a difference in MSE and accuracy values but it is not significant

4.2.2 Variables selection for the *rain* response

Similarly, the main idea is to build a linear model based on the selected variables as before, using the BIC, Cp-Mallow and R^2 score. If we consider each criterion, we are able to retain 7, 4 and 2 variables for R^2, Cp and BIC criterion respectively.

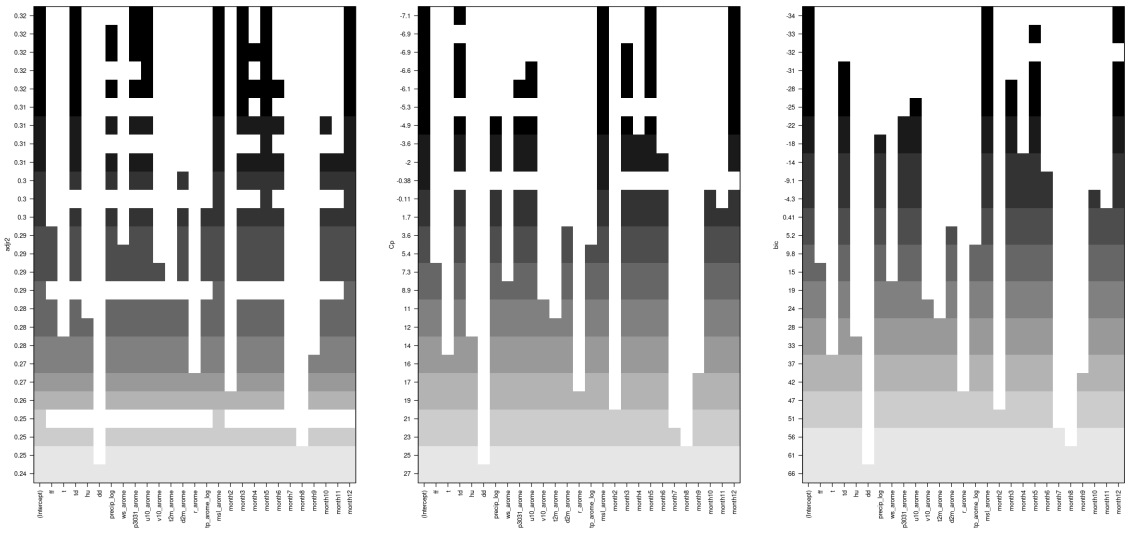


Figure 12: Variables selection by Cp, BIC and R^2 respectively

In the following, we build 3 different models for each of the 3 criterions and we compare them.

Linear model of selected variables by R^2
The model is well described as followings:

$$rain \sim td + p3031\_arome + u10\_arome + msl\_arome + month3 + month5 + month12$$

Training error for classification

- MSE: 15.791
- MAE: 2.414
- MAPE: 4610317.841
- R^2: 0.201
- Accuracy: 0.471

Prediction	Observation		
	high_rain	low_rain	no_rain
high_rain	121	97	49
low_rain	33	125	99
no_rain	1	12	13

Testing error for classification

- MSE: 19.383
- MAE: 2.531
- MAPE: 6147614.699
- R^2: 0.254

- Accuracy: 0.478

Prediction	Observation		
	high_rain	low_rain	no_rain
high_rain	30	24	13
low_rain	2	34	32
no_rain	1	0	2

Linear model of selected variables by Cp
The model is well described as follows:

$$rain \sim td + msl\_arome + month5 + month12$$

Testing error for classification

Prediction	Observation		
	high_rain	low_rain	no_rain
high_rain	31	25	20
low_rain	1	32	26
no_rain	1	1	1

- MAE: 2.540
- MSE: 19.690
- MAPE: 6912301.561
- Accuracy: 0.464
- R^2: 0.242

Linear model of selected variables by BIC
The model is well described as followings:

$$rain \sim msl\_arome + month12$$

Testing error for classification

Prediction	Observation		
	high_rain	low_rain	no_rain
high_rain	30	25	21
low_rain	3	33	24
no_rain	0	0	2

- MSE: 19.425
- MAE: 2.532
- MAPE: 6395443.655
- Accuracy: 0.471
- R^2: 0.252

Interpretation and Comparison: Comparing these 3 models, we realize that the simplest model with 2 variables seems to be the most efficient. Yet no significant difference is noticed. However, while comparing to the models built with `rain_log` response, both MSE and accuracy values show that these 3 models have worse performance. Moreover, we remark that in the class `no_rain`, there aren't too many predictions.

4.3 Linear regression with penalty - Rigde, Lasso and Elastic-Net

Penalized regression methods keep all the predictor variables in the model but regularize the regression coefficients by shrinking them towards zero. If the amount of shrinkage is large enough, these methods can also perform variable selection by shrinking some coefficients to zero. In this section, we will discover 3 common methods: Lasso, Ridge and Elastic-Net. Next, we will give more details of each method and the results obtained

4.3.1 Ridge regression

Ridge regression shrinks the regression coefficients, so that variables, with minor contribution to the outcome, have their coefficients close to zero. The ridge coefficients minimize a penalized residual sum of squares:

$$\min_{w} \quad \|Xw - y\|_2^2 + \alpha\|w\|_2 \tag{1}$$

The larger the value of complexity parameter α , the greater the amount of shrinkage and thus the coefficients become more robust to collinearity. Thus, Ridge regression is a linear regression method using ℓ_2-norm regularisation.

We choose the best model by choosing the best value of parameter α. Here, using the cross-validation through `GridSearchCV` with 5-fold, let's take $\alpha = 72.864$.

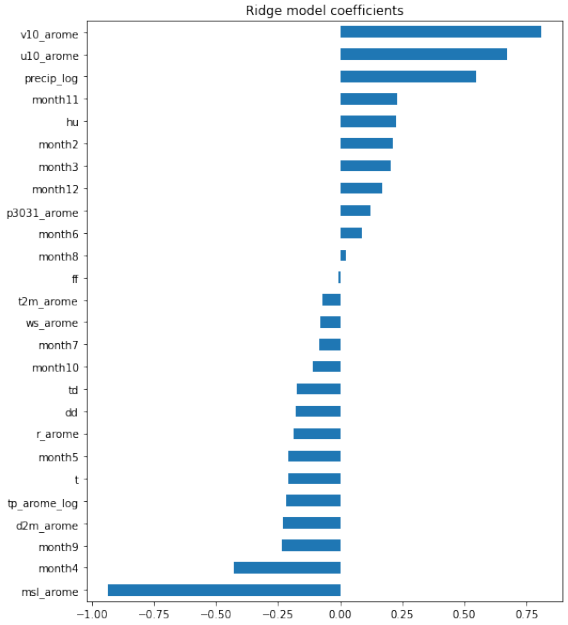


Figure 13: Ridge regression by variable `rain`

By observing this graphics, we do not delete any variables because there is no coefficient being equal to 0; however, these coefficients have values close to zero. Once the model is fitted with Ridge coefficients, we do some predictions on test sample and we obtain the results as follows:

- MSE: 20.265

- R^2: 0.220
- Accuracy score: 0.449

Now, we try to do similarly for the `rain_log` response. Using cross-validation, we can find the optimised $\alpha = 44.724$. We retain all variables but most of them have coefficients very close to 0. Once the linear model is fitted, we obtain the following results:

- MSE: 19.315
- R^2: 0.256
- Accuracy score: 0.522

Interpretation: We are still not disappointed with the results `rain_log` brings

4.3.2 Lasso regression

Lasso stands for Least Absolute Shrinkage and Selection Operator. It shrinks the regression coefficients towards zero by penalizing the regression model with a penalty term called L1-norm, which is the sum of the absolute coefficients. Mathematically, it consists of a linear model with an added regularization term. The objective function to minimize is:

$$\min_{w} \quad \frac{1}{2n_{samples}} \|Xw - y \ \|_2^2 + \alpha \|w\|_1 \tag{2}$$

Hence, the lasso estimate resolves this optimisation problem, where α is a constant and $\|w\|_1$ is the ℓ^2 norm.

We can choose the best model by choosing a good value of α. Here, the cross-validation is used to find the optimal $\alpha = 0.131$.

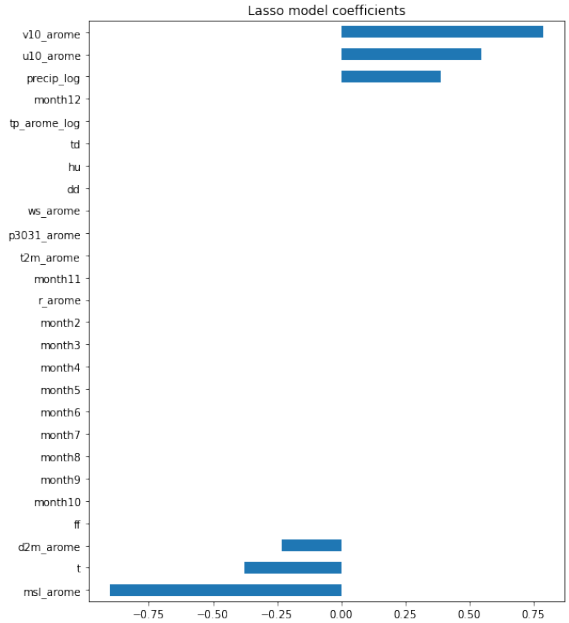


Figure 14: Lasso regression by variable `rain`

On this graph, we notice as well that in the Lasso model, 6 variables are retained and 20 other variables whose coefficient is 0 are released. These 6 variables `v10_arome, u10_arome, precip_log, d2m_arome, t, msl_arome` are used to fit a linear model for `rain` response. Once the model is fitted, we obtain the results as follows:

- MSE= 20.128
- R^2 = 0.225
- MAPE: 2750459238891696.0
- Accuracy score = 0.449

Now, we try to fit a Lasso model for the `rain_log` response. Similarly, the first step is to optimise the α value - penalty parameter. Using cross-validation, we can find the optimal $\alpha = 0.03015$. We retain 14 variables and delete 12 others. Once the linear model is fitted, we have the results as follows:

- MSE = 19.965
- R^2= 0.231
- Accuracy score = 0.5

Interpretation: The error and score values figure out the better results with `rain_log`.

4.3.3 Elastic-Net regression

Elastic Net produces a regression model that is penalized with both the ℓ^1-norm and ℓ^2-norm regularization of the coefficients. The consequence of this is to effectively shrink coefficients (like in ridge regression) and to set some coefficients to zero (as in LASSO). We control the convex combination of ℓ^1 and ℓ^2 using the `l1_ratio` parameter. The objective function to minimize in this case is:

$$\min_{w} \quad \frac{1}{2n_{samples}} \| Xw - y \ \|_2^2 + \alpha\rho\|w\|_1 + \frac{\alpha(1-\rho)}{2}\|w\|_2^2 \tag{3}$$

We can choose the best model by choosing a good value of α and ρ. Here, again, we use cross-validation to find optimal $\alpha = 0.122$ and $\rho = 0.958$.

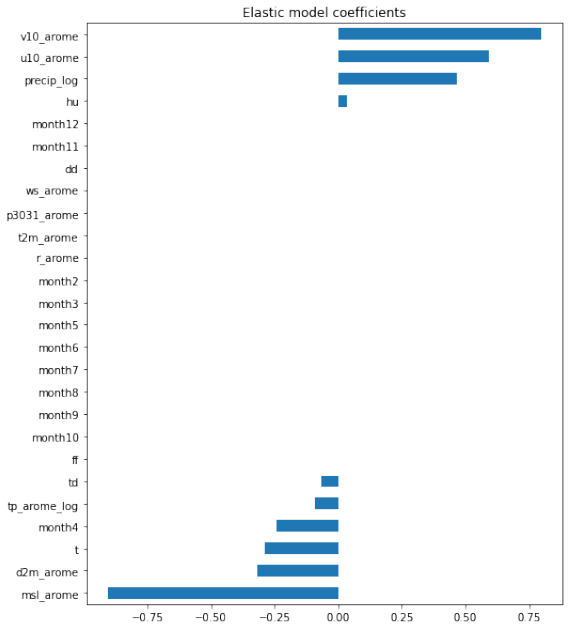


Figure 15: Elastic-Net regression by variable `rain`

According to the graph, in the current Elastic Net model, 10 variables are retained and 16 other variables whose coefficient is 0 are released. These 10 variables - `v10_arome, u10_arome, precip_log, hu, td, tp_arome_log, month4, d2m_arome, t, msl_arome`- are used to fit a linear model for `rain` response. Once the model is fitted, we obtain the results as follows:

- MSE= 20.129
- R^2= 0.224
- Accuracy score= 0.449

Now, we try to fit an Elastic-Net model for the `rain_log` response. Similarly, the first step is to optimise the penalty parameter. Using cross-validation, we can find optimal $alpha = 0.04082$ and $\rho = 0.6667$. We retain 21 variables and delete 5 others. Once the linear model is fitted, we obtain the following results:

- MSE= 19.952
- R^2= 0.231
- Accuracy score= 0.5

Interpretation: Just like before, using `rain_log` response gives us better results.

4.3.4 Comparison among Ridge, Lasso and Elastic-Net regression

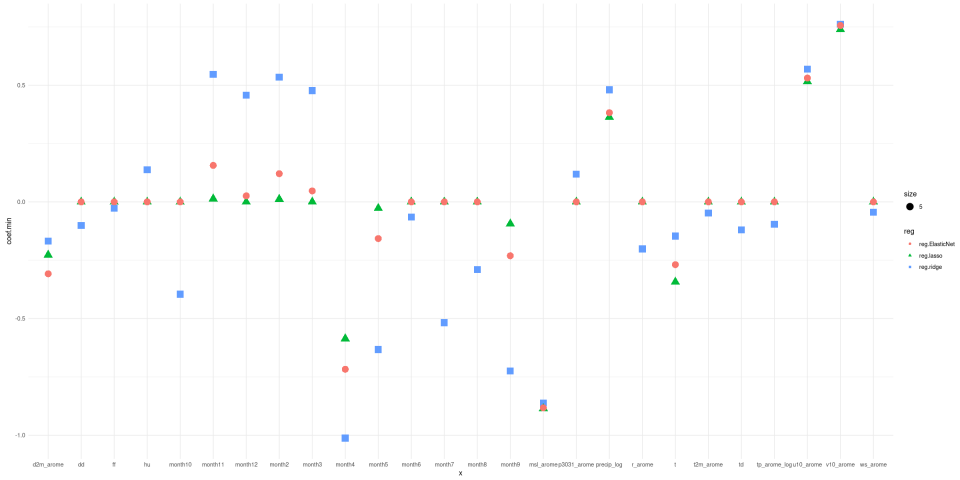


Figure 16: Coefficients of 3 models: Ridge, Lasso and Elastic-Net

According to the graph 16 of coefficients of the three models, we can observe clearly the properties of each penalty. Ridge regression with ℓ_2 norm regularisation, has coefficients close to zero. Lasso regression with ℓ_1 norm regularisation, also shrinks the coefficients toward zero. Whereas Elastic-Net regression combines these two properties by using both the ℓ_1 and ℓ_2 norm regularisation.

Moreover, we can realise that some variables have coefficients very close to zero with all the 3 models, which proves that they are the least important variables. For instance: `ff, month6, p3031_arome, t2m_arome, td, tp_arome, ws_arome,...`. Meanwhile, some variables prove important with their high coefficients, such as `msl_arome, precip_log, v10_arome, u10_arome,...`

As mentioned before, using `rain_log` response gives us a better results. When comparing these models by the errors and scores of models, we can see that with regularisation, we improve the result of fitted model and prediction than with the simple linear model. In particular, Ridge regression outperforms the two other models with higher R^2 score and accuracy as well as lower Mean square error. Lasso and Elastic-Net regression give nearly the same results.

4.4 Generalized Linear Models (GLM)

Generalized Linear Models (GLM) extend linear regression. First, the predicted values $\hat{y}$ that are linked to a linear combination of the input variables X via an inverse link function as follows:

$$\hat{y}(w, X) = h(Xw)$$

.

Moreover, in some new implementations (see Sklearn User's Guide), the squared loss function is replaced by the unit deviance. Hence, the optimisation problem becomes:

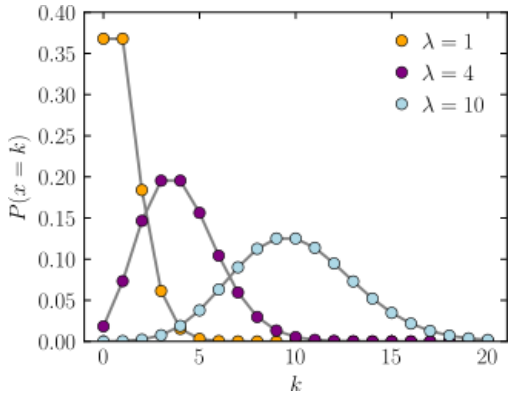
$$\min_{w} \frac{1}{2n} d(y_i, \hat{y}_i)$$

Some penalty terms (e.g. ℓ_1, ℓ_2 norm of w) might be added in order to regularize the problem.

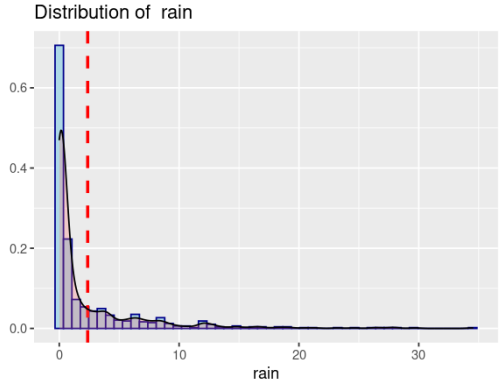
Since our response variable values `rain` are **positive valued**, we might try a Poisson regression with a log-link function, or Gamma deviance with log-link (or even higher variance powers of the **Tweedie family**

4.4.1 Poisson regression

We remark that the distribution of the `rain` variable seems similar to Poisson distribution. Hence, in this section, we try to fit a simple Poisson regression to see if it gives us a better result.



(a) PDFs of Poisson distributions (Source: Wikipedia)



(b) Histogram of variable `rain`

The unit deviance of the Poisson distribution is

$$d(y, \hat{y}) = 2(y \log \frac{y}{\hat{y}} - y + \hat{y})$$

After fitting a model, we obtain the following performance on the test set:

- MSE: 19.13
- MAE: 2.53
- MAPE : 2402814485053989.5
- R^2 - Score : 0.26

Using the given thresholds, we transformed the regression results into classification predictions. The performance on the test set is reported as below

Prediction	Observation		
	high_rain	low_rain	no_rain
high_rain	26	7	0
low_rain	25	33	0
no_rain	15	32	0

We can see that the algorithm can not detect the `no_rain` class. This is because the `no_rain` class is produced using the zero threshold, but in regression context, it's impossible to get a `rain` prediction to be zero.

4.4.2 Poisson regression with penalty

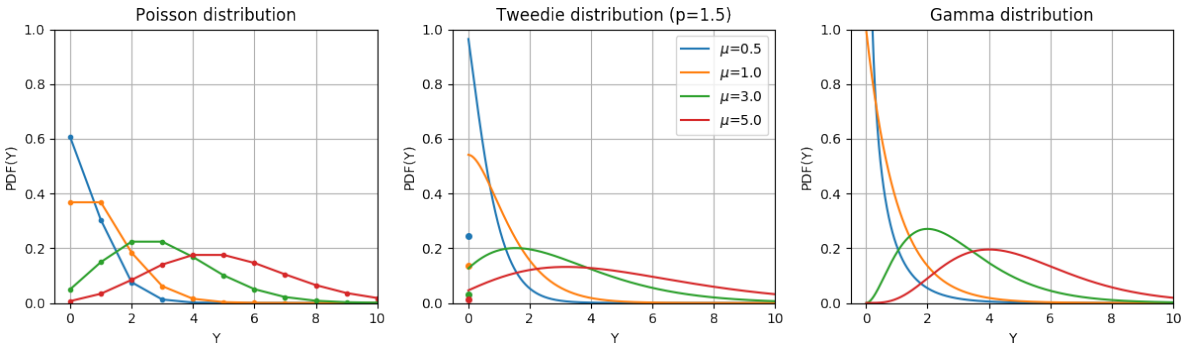
In this section, we used the same configuration as previously, except that we added a ℓ_2 norm of parameters in our optimisation problem. The penalty strength is tuned using cross-validation.
The performance on the test set is:

- MSE: 18.09
- MAE: 2.42
- MAPE : 2560581532952368.0
- R^2 - Score : 0.3

In regression metrics, this penalty seems to be working better.

4.4.3 Gamma and Tweedie regression

As mentioned above, since the response variable is positive, we fitted in this section the Gamma Regression and Tweedie Regression, since these distributions seem to be close to the one of `rain` variable.



Both of these regression include a penalty term (ℓ_2 regularization), and the strength of penalty and the `power` parameter of Tweedie regression is calibrated using cross-validation. The best results of these models are reported below:

Metric	Gamma Regression	Tweedie Regression
MSE	18.12	17.52
MAE	2.38	2.32
MAPE	2241305606891828.5	2306044107209599.0
R^2 Score	0.3	0.33

We can see that, in regression metrics, the Tweedie Regression seems to work better than Gamma, and even than Poisson. Unfortunately, when we use the given thresholds, **these models are not capable to detect the `no_rain` class.** This is because, the response variable is positive and it's set to `no_rain` only when it's equal to zero. But, in fact, these models are those of a continuous distribution. Therefore, its probability for being equal to a singleton value is zero. To remedy to this problem, we can allow some perturbations, i.e, we can set the prediction to be `no_rain` if the regression result is smaller than some `epsilon` constant (`epsilon` is small enough to be convenient with our data). More details can be found in the Python Notebook.

4.5 Comparison of the performance of regression models

In this section, we compare the performance of different methods presented above, using both regression metrics and classification metrics.

Regression Performance The regression performances on the test set using different metrics of all methods presented in this section are plotted on the following graph (note that the MAPE is plotted in **log-scale** since its values are very large). In this regression context, we aim to find a model **minimizing the test error** (MSE, MAE or MAPE) and **maximizing the quality of regression** (R^2 Score).

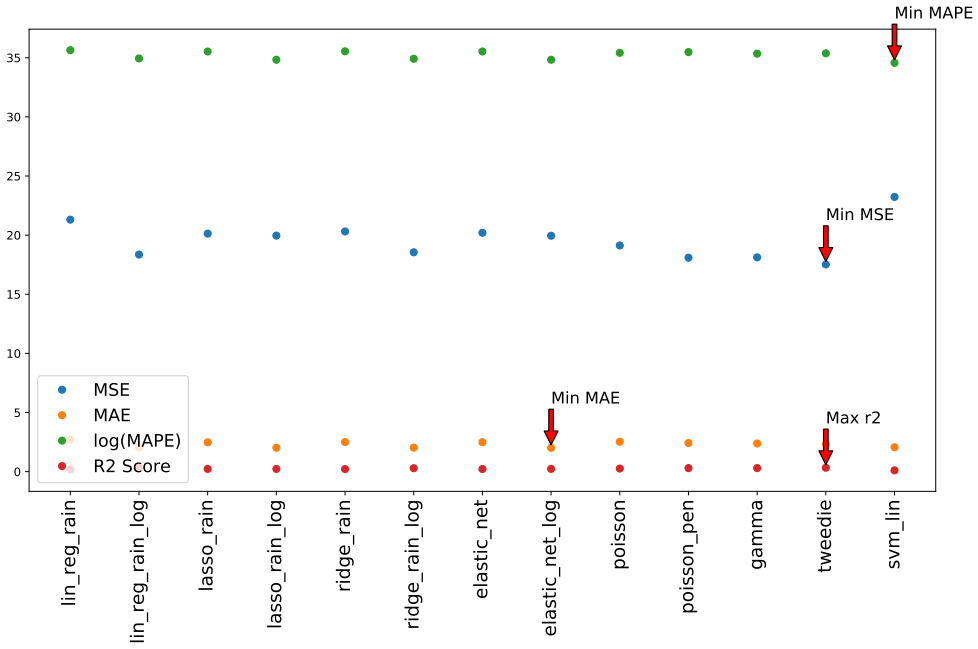


Figure 18: The performances of regression models

When it comes to linear models, we obtain a better result of linear regression with penalty terms than without penalisation. By choosing a good penalty parameter thanks to cross-validation, we can maximize the performance of the method and at the same time avoid overfitting. However, in general, the linear models are less efficient than non-linear models. As stated before, this is completely reasonable due to the non-linearity property of our data.

According to the graph 18, the usual Mean Absolute Error (MAE) is minimized with the Elastic-Net, which is a harmonisation of LASSO and Ridge. The MAPE is minimized with the Linear SVM, and this metric is more robust when dealing with outliers. But since in our data, there are some zero values in the response variable `rain`, we are dividing by zeros calculate the MAPE, thus this metric takes very large values (which was explained previously). Therefore, this is just a metric to have an insight of how the different metrics measure the quality of a model. Finally, it seems that the **Tweedie Regression** (with a ℓ_2 penalty) work very well in this regression context, it minimize the MSE and also maximize the R^2-score, its MAE is also close to the minimum.

Classification Performance We used the given thresholds to solve the classification problem and the performance on the test set is presented below:

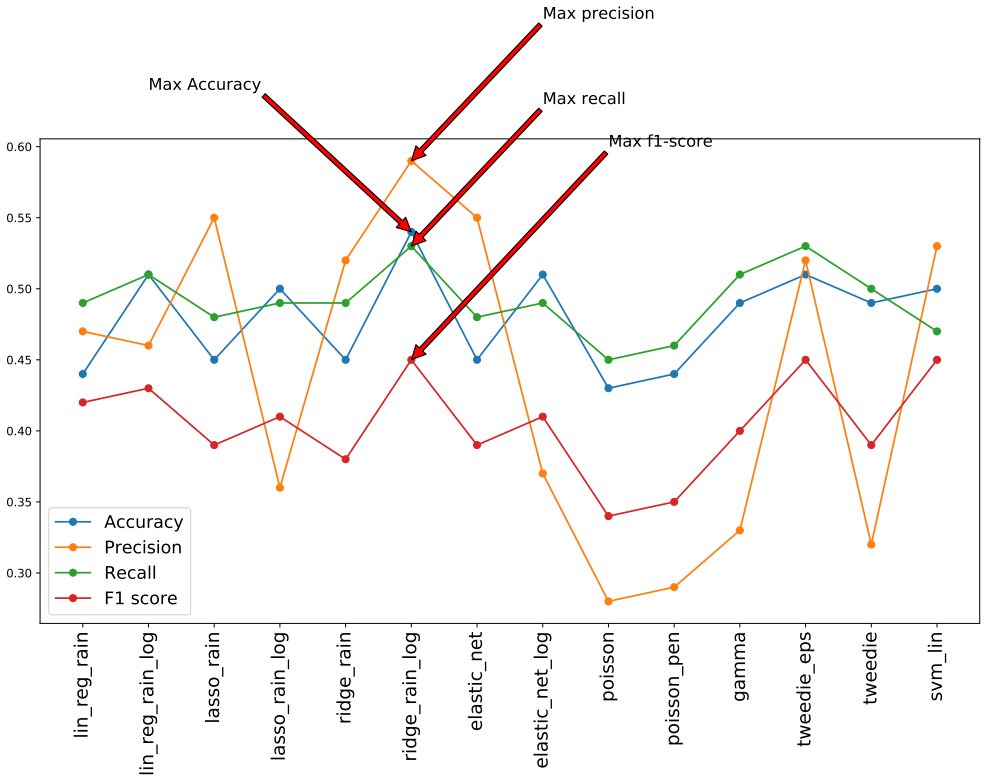


Figure 19: The performances in classification using regression models

The first remark is that none of the best models in regression problem give a good performance on the classification problem. The **Ridge Regression applying to `rain_log`** is the best model that maximizes all of the 4 metrics, with a precision around 58%. Besides, the Tweedie and Linear SVM gave an acceptable performance, both on regression and classification. The Poisson model seems to be the worse model in this classification context. All other models do not differ much in terms of classification performance.

As mentioned above, most of our models performs poorly when it comes to classifying the `no_rain` class, even though they do fine in the regression context. Again, setting the threshold for `no_rain` class to zeros might be the main reason.

5 Comparison between direct classification and "regression plus thresholding"

Here is a resume of our results for classification and regression models. Each model is evaluated by some of the metrics mentioned in the previous parts. The table 1 gives the performance of all models used in our project in the test set, with different metrics. For each metrics, the best score is highlighted in the red color.

		Regression Metrics				Classification Metrics			
Type	Model	MSE	MAE	MAPE	R^2_Score	Accuracy	Avg_precision	Avg_recall	Avg_f1_score
Regression Models	lin_reg_rain	21.31	2.71	3012×10^{12}	0.18	0.44	0.47	0.49	0.42
	lin_reg_rain_log	18.36	2.06	1494×10^{12}	0.29	0.51	0.46	0.51	0.43
	lasso_rain	20.13	2.48	2685×10^{12}	0.23	0.45	0.55	0.48	0.39
	lasso_rain_log	19.96	2.02	1342×10^{12}	0.23	0.50	0.36	0.49	0.41
	ridge_rain	20.31	2.5	2734×10^{12}	0.22	0.45	0.52	0.49	0.38
	ridge_rain_log	18.55	2.03	1456×10^{12}	0.29	0.54	0.59	0.53	0.45
	elastic_net	20.2	2.49	2695×10^{12}	0.22	0.45	0.55	0.48	0.39
	elastic_net_log	19.95	2.01	1340×10^{12}	0.23	0.51	0.37	0.49	0.41
	poisson	19.13	2.53	2402×10^{12}	0.26	0.43	0.28	0.45	0.34
	poisson_pen	18.09	2.42	2560×10^{12}	0.3	0.44	0.29	0.46	0.35
	gamma	18.12	2.38	2241×10^{12}	0.3	0.49	0.33	0.51	0.40
	tweedie_eps	17.52	2.32	2306×10^{12}	0.33	0.51	0.52	0.53	0.45
	tweedie	17.52	2.32	2306×10^{12}	0.33	0.49	0.32	0.50	0.39
	svm_lin	23.23	2.06	1033×10^{12}	0.11	0.50	0.53	0.47	0.45
Classification Models	k_nn					0.51	0.51	0.51	0.49
	decision_tree					0.49	0.51	0.46	0.47
	random_forest					0.49	0.51	0.46	0.47
	svm_linear					0.57	0.58	0.59	0.57
	svm_poly					0.51	0.51	0.51	0.50
	svm_radial					0.51	0.52	0.51	0.49
	svm_sigmoid					0.52	0.52	0.53	0.47
	one_hid_lay_nn					0.53	0.53	0.55	0.53
	mlp_nn					0.57	0.59	0.57	0.54

Table 1: Report of Model Performances

For the classification problem, according to the table 1, the scores of classification models oscillate around 0.5. It looks that the linear SVM and multi-layer neural network give the best results, with a performance approximating 0.6. Both of them are direct classification methods. Hence, in this case the direct classification seems to be better due to the better score, lower error and more simple use.

In the regression part, thanks to `rain_log` response variable, the performance of some models improved. In general, the penalized models prevent the overfitting phenomenon from occurring and therefore improve the performance of models. By looking at the distribution of the response variable `rain`, the generalized linear model with Poisson, Gamma and Tweedie distributions gave better results comparing to other linear models.

However, when twisting into the classification problem using the result from regression models, the prediction is not as good as expected. It would be harder to detect the class `no_rain` because the regression output is in some sens continuous and valued on $\mathbf{R}_{\geqslant 0}$, and we put it in `no_rain` only if this predicted value is equal to 0. Hence, the probability of having 0 is extremely low, that's why we seen bad performance on `no_rain` class and this leads to a bad average performance.

Conclusion

In this section we give some important points and comments of what we have done during the whole project.

Data Exploration :

- **Outliers**: for `precip`, `tp_arome`, and `rain` variables there are a lot of outliers for high values. To reduce this effect, we have performed $\log(\cdot + 1)$ transformation.
- **Variable pertinence**:
  - **Multi-dimensional Analysis**: variables t and td are both correlated with variables $t2m\_arome$ and $d2m\_arome$. On the other hand, the most influencial feature on our response variable $rain$ is msl_arome.
  - **Principal Component Analysis (PCA)**: After performing PCA we kept 6 variables. They represent respectively, precipitation, temperature informations, wind direction, humidity and wind speed.

Data versus Model: We have originally 15 explicative variables to explain the `rain` (and `rain_class`) response. But we have only 688 individuals in the dataset and therefore only 550 individuals in the training set, living in the feature space of dimension 15. We can say that the amount of data was not enough to explain the response variable in a space of dimension 15. But it may be possible that our chosen models were not good enough to be able to generalize to the training set. These might be the reasons explaining why we did not get a better result.

Data imbalance: Here we have a small but imbalanced dataset (in terms of classes). We have $292, 208$ and 188 individuals in the `high_rain`, `low_rain` and `no_rain` class respectively, which corresponds to $42, 30,$ and 28 percents of data respectively. This is also a possible reason explaining why we had difficulty to detect the `no_rain` class.

Thresholding: Assigning an individual to `no_rain` class if the `rain` is equal to zero is actually realistic, but yet, in our problem, it seems to be a bad idea, especially when we use regression model for classification purpose. Our models have difficulty to detect to `no_rain` class leading to a bad average performance.

Classification: In this section, we tried to use the direct classification algorithms. Generally, the accuracy score is not as high as we wanted to be. However, we have to agree that SVM with linear kernel gives the best prediction result. In fact, the other kernels of SVM work well, but only for detecting `low_rain` and `high_rain` classes, and they have the same problem with `no_rain`. Besides, the multi-layer neural network proved its performance with a very close score to the one provided by the SVM with linear kernel.

Regression: In this section, we tried to test many methods to fit models for prediction purpose. As demonstrated, we see clearly the non-linearity property of our data. Thus, it is not linearly separated. Therefore, it is normal that we weren't able to obtain good results by using the linear models, even though we tried to improve it by using `rain_log` response as well as adding the regularisation terms. Meanwhile, generalized linear models (Poisson, Gamma and Tweedie) showed better performance, especially in the case of Tweedie regression since the distribution of the response `rain` was very similar to the Tweedie distribution. In general, we got a good performance in the regression context.

Solving classification problem with regression: Although the regression predictions in the previous section weren't bad, we can not obtain very good classification results as we expected. The model giving the best classification prediction is Ridge regression with `rain_log` before twisting to the classification. Still, this result is not better than the direct classification method. The main reason is the strict condition of class division. The probability of belonging to class `no_rain` is much lower than the two others. Hence, it would be better if we used the direct classification methods in this case.

What we have learned during this project: During this project, we learned and understood the concept of many algorithms in Machine Learning, how a Machine Learning project is designed and built upon real world data. By completing this project, our theoretical understanding about Machine Learning has improved significantly, and we also acquired a lot of technical skills.

Acknowledgement: We would sincerely love to give a big thanks to our supervisor - Mme. Béatrice LAURENT for her advice and support through the courses and especially this project. She provided us with high quality knowledge about Machine Learning, whether on the theoretical concepts or the applied aspects of the subject. We highly appreciated this occasion of working with her!