Introduction

The task of this project was to predict whether it rains in Pully on the next day based on data from other weather stations in Switzerland. In this report, we describe the workflow used to achieve this task.

Our work was composed of 4 parts. Firstly, we explored the data. The goal was to understand and to clean the training data. Then we build several prediction models. We divide the models into two categories, the linear and the non-linear models. Finally, we ran an evaluation process on those models to be able to choose the model with the best performances.

Data explorations

All the data explorations were done using the script **dataVisualitation.jl**. The training set is composed of 528 predictors and 3176 data points. We begin the data exploration by simply plotting some variables and computing some general statistics on the data. What we found was that, firstly, the training set contains quite a lot of missing data ($\tilde{5}0\%$ of the training points contain a missing value for at least one predictor). This result shows us that removing all the missing values would reduce strongly the size of the training set. Secondly, by plotting some variables, we saw that the training data were ordered by time. Knowing it, we knew that the training datasets would have to be shuffled before being used to fit models. Thirdly, we saw that the training set has an equivalent number of true and false labeled data points (56% False, 44% True).

In the second part, we tried to understand the correlation between the different predictors. For doing it, we compute the Pearson correlation matrix between each normalized predictor. We visualize this matrix using a heatmap. This analysis did not show any significant correlation between pairs of predictors. Then we compute the correlation between each predictor and the response variables. What we found was that most predictors have a low correlation with the response variable. Based on those correlations analysis, we concluded that there was not enough evidence to remove some predictors.

The last part of the data exploration was to perform a principal component analysis on standardized data set. We plot the results of PCA1 to PCA3. The results of the principal component analysis did not show any clustering that was induced by the label of the data. We also observed that 96 PCs explained 90% of the variance of the data.

We can summarize the results obtained during the data explorations:

- The number of predictors is quite high in comparison to the number of data points.
- The training data needs to be shuffled before being used in a model.
- Filling the data could be necessary to obtain better model performance.
- No predictors could be removed based on the performed statistical analysis.

We choose to apply the following procedure for the project. We would feed the models with three different training sets: the cleaned data (removing all the missing data), the standardized data (removing all the missing data and performing a standardization on this data), and finally the filled data (replace the missing data with the mean value of the predictors). All the data sets would be shuffled randomly before being used.

Workflow

In this project, we tried to follow a specific workflow to create and evaluate the models. Firstly, we imported the three types of training sets. Then, we train the different models on those data. We mainly use the $MLJ.TuneModel$ for finding the optimal hyperparameters. This method builds a model using several values for each hyperparameter and chooses the best model based on the performance of cross-validation. When we had a fitted model, we saved it into a separate file. Then, we performed the evaluation in a specific script (**evaluation.jl**). For the evaluation and optimization of the models, we used the area under the curve (AUC) as a performance parameter. When we were happy with a model, we generated a submission with the **submissionModels.jl** script. This workflow allows us to get reproducible and understandable results.

Linear models

All the exploration of the linear models was done using the **linearModels.jl** script. For the evaluation, we used a 10 K cross-validation with all the models. We mainly focus on the logistic classifier. Firstly, we were interested to see how the feeding data and the regularization impact the performance of the models.

We began to produce a simple logistic classifier with no regularization and trained on the cleaned data. Then we used the lasso and ridge methods on the logistic classifier to see the impact of the regularization. We saw that the simple logistic classifier and the one with L2 regularization gave us some decent results for

a first benchmark (No regularization, AUC: 0.86, Ridge, AUC: 0.91). Based on those results, we decided to rerun the experiments, but with the subset of data containing only weather points for the Pully station. This time all the models performed poorly (No regularization, AUC: 0.87, Ridge, AUC: 0.87, Lasso, AUC: 0.87). Finally, we decided to refit the logistic classifier with the Ridge regularization on the filed data. This model gave the best results for linear models (AUC: 0.91). From the linear models, we saw that the number of data points was important and that the regularization improve a lot the performance of our model. On Kaggle, we made two submissions for the linear models (Logistic classifier with ridge regularization on the cleaned and on the filled data). Those submissions could be reproduced with the **submissionModels.jl** script($model_3$ and $model_{11}$).

Non-linear models

The implementations of the non-linear methods were done with the **otherModels.jl** and the **neuronalModels.jl** scripts. We tried several methods, and again we used cross-validation for the evaluation. To reduce the computation time, we choose to reduce the cross-validation from 10 to 4. We begin with the k-nearest neighbors model. We implemented three versions, the first one was fitted with the cleaned data, the second one was fitted with the pully station data points and the third one was fitted with the filled data. The performances of those models were quite low (KNN cleaned data, AUC: 0.90, KNN filled data, AUC: 0.90, KNN Pully subset, AUC: 0.90). We submitted the predictions of the two best models on Kaggle ($model_1$ and $model_{12}$)

Then, we tried to implement some neuronal networks. Since neuronal networks are sensitive to normalization, we run all our models on standardized data. We tried two versions of multilayer perceptrons. The first one was a self-tuned model and for the second one, we manually tune the hyperparameter to reduce the computation time. The neuronal models performed quite badly (AUC: 0.52). Then we tried to fit those model with the non-standerdized data, this time we get better results (AUC: 0.87). We submit the best model on Kaggle ($model_4$).

Finally, we tried three different tree bases methods (XGBBoost, Decision Tree, Random Forest). Those methods are quite simple to implement and do not require normalization of the data. For each method, we used a self-tune model. In the first approach, we ran all those models with the cleaned data sets. Except for the tree decision model, the two other methods obtained AUC values that were above 0.90 (XGBBoost, AUC: 0.92, RandomForest, AUC: 0.91). Based on the fact that we saw an improvement of the linear models with the filled data, we choose to rerun the two best models on the filled data.This procedure only improved the performance of the XGBBoost model (XGBBoost, AUC: 0.93). We generated a submission with this model on kaggle ($model_6$).

Based on the results obtained, we decided to focus on the XGBBoost model. What is important to understand is that XGBBoost has three main parameters (learning rate, number of rounds, and maximal tree depth). We try several combinations for those parameters and did not find any massive improvement on the model. We also tried to feed the XGBBoost models with standardized data, again no improvement was made. Then, we tried to do what is called data augmentation. The idea behind this is to increase the number of data points, thus being able to feed the algorithm with more data. Our data augmentation was done in the **dataVisualitation.jl** script. We generated synthetic point with a simple interpolation between two points. Each time that two consecutive points have the same label, we compute the linear interpolation between those two. This data augmentation allowed us to generate 1'637 points. We added those points to the original datasets and save them as a CSV (augmenteddata.csv). We ran the XGBBoost methods with the augmented data. The results obtained were similar to the one with the filled data. An important difference with this new model was the fact that the AUC was more stable across the K fold. We made a submission with this model on kaggle ($model_8$).

Conclusion

With the linear models, our best performance was obtained with the logistic classifier with ridge regression trained on the filled data (AUC: 0.91). For the non-linear models, our best performance was obtained with the XGBBoost model trained on the augmented data (AUC: 0.93). We choose to select the following submissions: XGBBoost model train on the filled data and the XGBBoost model train on the augmented data. We suspect that the XGBBoost model trained on the augmented data would perform better on the complete test dataset compare to the other one.

Based on the small improvement with the augmented data, we could conclude that, our model is close to the irreducible error that is induced by the noise of the data. For further improvement, we could try to spend time tuning a multilayer perceptron. In theory, the neuronal networks should give us slightly better results.