Mini Project BIO-322

Team: L'EPFL c'est de l'eau

Introduction

Our goal was to predict whether or not there will be precipitation the next day in Pully based on data we obtained from twenty-two various stations around Switzerland measuring six different quantities at four time periods.

Preliminary results

1. Exploration and visualisation of the data

The input data consists of different measurements with some variables having a wider range of values, whilst others have positive and negative values. On the other hand, the output is of type bool, being true if it rains or false if it does not.

In order to get a feeling for the data, we visualised various correlation plots. When the data is highly correlated, the plots are dark blue and go into yellow and red the smaller their degree of correlation is. For example, we can also observe for Pully that the sunshine, air temperature and the radiation are strongly correlated, which makes sense. On the contrary, delta_pressure and radiation are very weakly correlated. In general, we noticed that the measured quantities can have a certain degree of correlation across different stations (radiation, air temperature, delta pressure) or within the same station at different times (radiation in ABO). However, this does not mean there is much correlation with precipitation on the following day in Pully. Therefore, we chose not to neglect any data and train our machines on the whole data set we received.

2. Transformation of the data

Firstly, in order to work with the output, we coerced it to a multiclass. Next, to handle the missing data, we transform the training and test set inputs using the FillImputer() function. Even though these are not "correct" data, we chose not to use dropmissing() as it would have removed around half of the data meaning we would have less points on which to fit the machine making it less precise. We finished by standardizing our data since the variables have varying scales as they are the measurements of different quantities.

Results : finding the best models

In both the linear and non-linear methods, our workflow was similar. We first randomly split the data into a training and validation set with a function. We then fit the training data on a machine and viewed the confusion matrix. If we thought it was interesting, we then created a self-tuning machine with chosen hyperparameters to tune. We then evaluated the best model on the validation set. If this result was satisfactory, we did the same process but on the full data set for a more specific fit and then used the machine on the test set. See “random_forest_manual_tuning.jl” for an example.

A linear method: Multiple logistic regression

Since we have a classification problem with multiple input variables, we chose a multiple logistic regression model as our linear method. In addition, we tuned the model with the lambda hyper-parameter from L2 regularization which penalizes the weights that are too big according to the mean of the data. We then compared with L1 regularization which looks at the median of the data. This made more sense as we realised some data were spread following a gaussian curve while others followed a negative exponential. This second regularization gave us much better results. To evaluate the model, we returned the confusion matrix and auc showing the number of correct and incorrect predictions the best model made. We had also tried applying PCA, but quickly realised it produced much worse results since it reduces the dimensionality and thus results in the loss of spatial information which is needed for our classification problem.

1. A non-linear method: from Random forest to XGBoost

When deciding which method to use, we chose not to do K-Nearest Neighbours as it assumed the idea of similarity and would thus not work for our data. As Random Forest is generally easier to fit, we started with that model, first making the machine ourselves with 500 branches, around the number required to compare to a neuronal network. We then fitted a self-tuning machine which iterated on the number of branches using cross-validation and the area under the curve to find the very best version. Our results were already much better, but we still thought there was room for improvement. So, instead of using this bootstrap technique that only averages the best results at the end of the process, we looked into XGBoost which we tuned on the maximal depth of the branches, the number of rounds and the learning rate. To optimize the parameter ranges, we returned the best model with the best kept parameters. This method has a higher predictive power and performance, but can quickly result in overfitting the training set if there is a lot of noise. For this reason, we did the cleaning beforehand and many cross validations when tuning the machine. However, running this model takes a lot of time.

Conclusion

Our best linear method ended up being logistic regression with lasso regularization on standardized training and test sets (logistic_regression_l1.jl) with the hyperparameter lambda = 5. This gave us an accuracy of 0.95487 on the test set. On the other hand, our best non-linear method, XGBoost, gave us a score of 0.96084 on the test set (XGBoost.jl). The best hyperparameters used were 900 for the number of rounds, a learning rate of 0.03 and a maximum depth tree of 5. There came a point where our results reached a certain threshold which could mean that we attained the irreducible error of the model. To go even further, if we had more time, we would also attempt to finely tune a Multiple Layer Perceptron which could give even better results.