Machine learning miniproject “will it rain tomorrow”

Picchi Evan, Louis-Alexandre Leger

December 2021

1 Introduction

The aim of this project was to predict whether it would rain tomorrow in Pully using +500 predictors. By training different models, we tried to get good approximations using the AUC of the ROC curve as a measure for the performance of the model.

2 Data cleaning and preprocessing

The first step of our project was to import the training set (downloaded from the kaggle competition page) and analyze it. We first applied the “describe(DataFrame)” method which showed us some properties for each predictor such as the mean, the min value & the number of missing values. To deal with the missing data from our training & test set we decided to use the “FillImputer” (fill missing data with median value) method because we saw that each column had a few missing values (for example, out of 3000 samples, the predictor with the most missing values was “STG_delta_pressure_3” with 327 missing values). We also tried to drop all of our missing data values, but this led to our data set size being reduced by half, so we considered it best to replace the missing data with the median value.
Next, we tried to apply the standardizer method to our training set (to which the missing data has already been added). We ignored the predictor ”:ALT_sunshine_4” for standardization as during data analysis, we saw that it only had 0 values for the standard deviation. Later on, we will ignore other predictor for standardization for hyperparameter tuning with cross validation. We then decided to plot the data, at first, we tried to plot a multiple scatter plot of 2 predictors (other than the precipitation_nextday) and color the points in function of precipitation_nextday. We plotted GVE_wind_direction_1 vs SMA_delta_pressure_4 and even if it is not easy and obvious, we notice that there are ”data” groups (positive rain data tend to be more in a certain region of our plots while ”negative” rain in another). However, in general, plotting the data was not really useful for us here due to the high number of predictors (which can imply very complex relationship with our classification output) so we used the plotting to observe that certain data & predictors seemed to follow a structure and others not.
We have also displayed the correlation matrix. This matrix justifies the use of regularization as we can see from this matrix that some predictors are correlated and the others are not.
Furthermore, we plotted the biplot (for the 2 first principal axes) & obtained it by performing Principal Component Analysis (PCA & with standardization) on the training data to see if we could find any interesting results, but this was also a complicated task to conclude due to the high number of predictors.
Our logical next step was to divide the training set provided by the teacher in two “subsets”, one training set (to train and find the best hyperparameters) and one test set, to estimate the AUC (the error measure we mainly used to evaluate the efficiency of our model, the area under the ROC curve). For this we used a random seed: 1234, and allocated 80% of our total data to a training set & 20% to our test set.
Also note that we always used parameter tuning with Cross validation (with $k = 10$ or$k = 5$)

3 Linear method

For our linear methods, we tried multiple variations of the multiple logistic regression : The first one being the **”classic” Logistic classifier** (so without L1/L2 and without standardization) and with this we obtained a training AUC of 0.972466 (and a misclassification rate= 0.0937008) and a “test” (on the test set from the initially splited data, see introduction) AUC of 0.904771 (and misclassification rate= 0.169811), the low misclassification rate shows that we overffit on the training set. (we also tried to train the ”classic” logistic classifier with standardization to see the difference and we got a slightly better result for the training AUC) Then, because of the large number of predictors (as well as the correlation matrix) we decided to do regularization : We performed hyperparameter tuning for 4 models, note that we trained this 4 model for comparison, but before running the hyperparameter tuning machine we expected to have better result for the standardization method because of the impact that different units can have on the predictors.

- Logistic classifier +L1 (without standardization) : the gradient method did not converge in 1000 iterations, so the best training and testing AUC are around 0.5 (which is the same for random values)
- Logistic classifier +L2 (without standardization) : we got a large $\lambda =$ parameter which was expected due to the high number of predictors
- Logistic classifier +L1 (with standardization) : the gradient method didn't converged in 1000 iterations, so the best training and testing AUC are around 0.5 (which is the same for random values)
- Logistic classifier +L2 (with standardization) : we got a large λ parameter but less large than the version without standardization, which was expected.

In conclusion, we can say that we obtained the best test AUC for The logistic L2 classifier with standardization :0.923591 (misclassification rate = 0.158805).Note that when we submitted it to kaggle we obtained an AUC of 0.95537. We thought this result could be improved using non-linear methods.

4 Non-linear methods

4.1 KNN classification

We first started with knn classification with hyperparameter tuning (on K) and obtained a test AUC of 0.917774 which is less than for the best linear model, so then because we know that KNN classification is sensitive to standardization we performed it with standardization and got test AUC of auc 0.918337 which is better but still does not exceed the best linear method.

4.2 Multilayer perceptron

The second nonlinear method we used was creating Neural networks, we first started to explore by hand a simple neural network, with one input layer, 1 hidden layer composed of 128 neurons and 1 output layer (and σ =sigmoid because we want a probability), we first got a test AUC = 0.915831 (We used seed= 1 to make the result reproductible, this seed was found by trying multiple seed) and we used it for the other neural network to compare them. Then, because we know that Neural networks are sensitive to standardization we performed it (with the same number of neurons, seed,...) and got a better result: a test AUC of 0.928875, BUT we also got a training AUC of 1 which could indicate that we overffit our training set. We also wanted to test neural network with multiple layers because we know that adding layers can work better than smaller: so we created a neural network with 3 layer of each 100 neurons and got a training AUC of 1 and a test AUC of 0.918488 (which is less good but this was expectud since we just tried it without hyperparameter tuning) Finally, we used the TunedModel to optimise the neural network model with 1 layer with multiple parameters : the dropout (which could help us not to overffit the training set), the number of neurons in the hidden layer and the number of epochs. We found a test AUC of 0.928633 and training AUC of 0.996759, (so this model was a bit less good than the one without parameter tuning, but on the other hand it is also possible that the training set is less overffit since the training AUC is less than 1. The fact that the test AUC was smaller can be explained by the fact that we could use more fine "grid" for the parameter to tune but for execution time reasons we decided to do the hyperparameter training without too many values). The above results did not really convince us, so we decided to use other methods, perhaps simpler (and faster) to optimise: Tree methods

4.3 Tree based methods

For tree-based methods, we first ran the TunedModel to find the parameter n_tree. We ran it with and without standardization and expected the two methods to give us similar results as the RandomTree method is not as impacted as neural networks are by standardization. Here we got slightly better results for the method with standardization with an AUC = 0.937178 (for the non standardized one we found AUC = 0.936997 which is not that different as expected). Note that although both results offer better results than the majority of the methods seen above, for both methods we obtained a training AUC of 1 which shows that there is a probable overffit of the training set. We then tried to use XGBoost package (gradient boosting framework for multiple languages) because it can usually get more efficient results for tree-methods. We started by optimising the model (without standardization for the hyperparameters : num_round,max_depth, η (learning rate)), with this model we got a test AUC = 0.946095 and training AUC of 1 (may also indicate an overffit of the training set, but because of the good test AUC we kept it), we decided to submit this result on kaggle (on the public board so for 30% of the true testing set) and got AUC= 0.95981 which is the best result we got on kaggle. Note that here (for reproducibility, we used the default seed, which is 0)

5 Conclusion

In conclusion, we tried to train several models and found interesting results, our working method was to first test different models "by hand" then to tune them and then observe and compare the results. The best result for linear method was found for multiple logistic regression with L2 and standardization, test AUC of 0.92359. The best result for nonlinear method was for the boosting gradient method (by using XGboost) and we found AUC = 0.946095