PREDICTING IF IT WILL RAIN TOMORROW IN PULLY

1 INTRODUCTION

In every day life, people look at and trust meteorological previsions. These predictions became even more accurate by using some data and training some machine learning methods. Through this project, we were interested in a similar and specific problem: to know whether it will rain the next day in Pully or not. To do so, we had a large data set containing some meteorological features for many different Swiss cities, as well as plenty of methods we learned about in the lecture. However, in the final version of our code, we only kept the linear and non-linear methods that gave us the best prediction results.

2 INSPECTION AND TRANSFORMATION OF THE DATA SET

The data set we used contained 6 different parameters that have been measured for 22 Swiss cities, every six hours during an entire day: the average solar radiation, the change of pressure within 6 hours, the average temperature (2 meters above the ground), the total sunshine duration over the last 6 hours and the average of wind speed and direction.
This gave us 528 different features and a last column mentioning whether it had rained in Pully the day after or not. As the measurements for each feature were done 3176 times, the final DataFrame built upon the training set we were given initially contained 529 columns and 3176 rows.

However, by exploring this DataFrame, we found out that there were some missing values. As performing DropMissing on those values lead to losing nearly half of the data, we decided to replace the missing values by the mean on the corresponding column. Then, we also saw that some values were extremely large or small. Those values making no sense in comparison to the rest of our data, we decided to set the values smaller (in absolute value) than 10e-8 to 0 and the ones larger than 10e3 (in absolute value) to the mean of the column (as there was few of them). Setting large values to more accurate ones prevents the shift of the data variance to a value that would be less representative of the overall data set.
Finally, two columns (:ZER_sunshine_1, :ALT_sunshine_4) only had zeros for each row. Therefore, those columns did not give us any useful information in order to tune an accurate model. In addition, those columns would have interfered with the standardization we then performed on the data, giving rise to NaN values in the standardized DataFrame. We thus dropped them both in the training data and in the test data. Obviously, all the other changes made to the training data have also been applied to the test data.

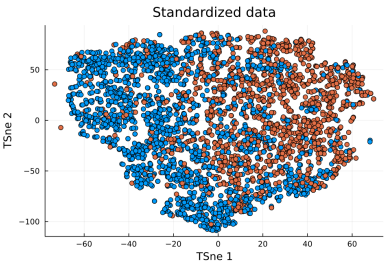


Figure 1: Representation of the data using TSne

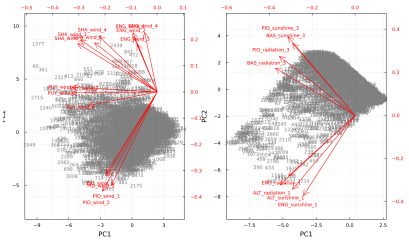
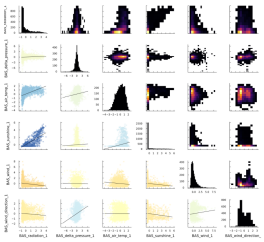


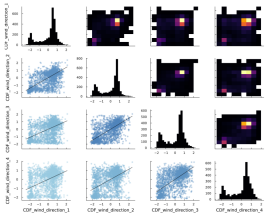
Figure 2: Wind speed correlation with time (left) and radiation/sunshine correlation(right)

To further inspect our data, we used PCA to see potential correlation between the different features. However, no global correlation behaviour has been seen. Therefore, we did a few correlation plots using some features from cities we randomly chose. This was done in order to have a global idea of the correlation between the different features in the data set. From the figure (a), we found out that for a particular time in a particular city, some features like the sunshine and radiation were correlated, but others like the wind direction and the wind speed were not. To further inspect the correlation between the radiation and sunshine, we used PCA on some of these features for given times and cites (figure 2) and found out that they were indeed correlated. Furthermore, some of these features also to be correlated between cities! Within a particular city, some features are correlated with time, like the wind speed (figure (b)), but others like the wind direction (figure (c)) are not.

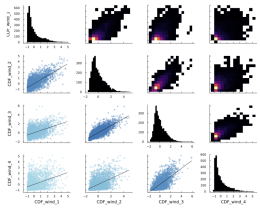
Finally, in order to reduce the large difference we could have among the different values of our data, we standardized the training data set. This is important in order to be able to compare the data, as, thanks to the standardization, they are scaled to a variance 1 and mean 0. This is even more important as we used regularization methods like ridge that works by penalizing large coefficients.
For the test data, as some other columns than the ones previously mentioned had all their values set to 0 (in the test data), we did the standardization on all columns but those.
By changing the missing values for the mean of the column, we might have added some noise in addition to the one already present.



(a) Correlation plots for BAS features at time 1



(b) Correlation plots for CDF wind direction over time



(c) Correlation plots for CDF wind speed over time

Therefore, we performed PCA to denoise the data sets with a proportion of variance explained of 0.99, giving us an output dimension of 306. To check that we did not remove too many components, we plotted the data in 2D before and after the denoising and saw that they looked similar. We thus only kept the 306 principal components. Finally, we also decided to plot the traning data with TSne to see if some clusters exists in the data. By looking at figure 1, we see that it might be the case, even though there does not seem to be any defined decision boundary.

3 MACHINE LEARNING METHODS

First of all, as the results are classified as true or false depending on whether it rains in Pully or not, choosing classification methods made sense for this project. We thus tried several of these methods: KNN & Multinomial classifiers did not give us good results (even when tuned for the KNN). A logistic classifier seemed like a better option given the results that we first obtained with it, but as we had a lot of data, we thought that regularizing them could result in even better predictions. Indeed, using regression in a logistic classifier penalizes some features, which can be interesting as we have a lot of them and still maybe some slight offsets. On the other hand, neural networks represent a very useful non-linear method as we can easily modulate their complexity and parameters. For all these reasons, we decided to keep a logistic classifier and a neural network classifier as our two final methods, on which we performed regularization.
We also used cross-validation with 10 folds during the tuning in order to try to prevent over-fitting on the training data and also used repeats for the same purpose.

3.1 LOGISTIC CLASSIFIER WITH ELASTIC NET REGRESSION

This supervised learning linear method, enables to predict the probability of a particular event to be in one of two classes, given an input. In addition to this, we used elastic net regularization (mix between lasso and ridge) in order to have the most complete model. Indeed, by doing so we cover all of the studied regularization types : the machine searches what Ridge/Lasso combination is the most accurate, even if it is only Lasso or Ridge in the end. In this model, we also tuned the lambda and fitted the machine on the standardized data. We did not use denoised data as we already performed regularization on them. To prevent over-fitting, we performed 2 repeats in addition to the regularization and cross-validation with 10 folds. After several tuning and having diminished the range size of the tuned parameters in order to be more precise, we finally found that the best model has a lambda of 115.556 and a gamma, which corresponds to the contribution of the lasso regularization, of 0.433333.

3.2 NEURAL NETWORK CLASSIFIER

This non-linear method is the one we chose for its flexibility and the large number of parameters to tune it has. Because of its high modularity, we used relu as activation function. The output will in any case return probabilities through the last neuron layer. In this model, we tuned the lambda and alpha values of the elastic net regularization. As neural networks can be very flexible, we tried to prevent over-fitting by setting the number of neurons to 32, the dropout to 0.5 and by doing some repeats. The dropout can be a good way to prevent over-fitting as it sets a probability for each neuron in the network (0.5 in our case) to be ignored during the training. In addition, we used 10 folds cross-validation and decided to set the batch size at 64, in order to have samples large enough to prevent over-fitting, but small enough to enable flexibility. Finally, we tuned manually the epochs number in a for loop, looking at the AUC measurement for each epoch we chose. We kept the model with the "lowest" epochs number giving the highest AUC, in order to prevent over-fitting. We ended up finding an epoch number of 33, and finally a lambda of 0.01675 and an alpha of 0.

4 CONCLUSIONS

For this project, we began by inspecting the data and removing the useless columns as well as transforming inaccurate or missing values by more accurate ones. As we had a lot of data and features, we decided to run standardization in order to be able to compare the data and do right treatment on them. To find the best models, we processed by measuring the AUC each model gives and kept the one that had the highest, while tuning different hyperparameters. We used elastic net regularization in the logistic classifier on standardized data and for the neural network, we used the data that were standardized and denoised. Over-fitting was a problem to deal with as we used flexible methods and in order to prevent it (or at least minimize it), we did cross-validation and decided to use several repeats, as well as adjusting the neuron numbers, batch-size and number of epochs.