Ramp Competition: Number of Air Passengers

Project Report

By Paul-Antoine Girard & Adrien Toulouse

Username: toulouse_girard
Final submission name: new_test_4

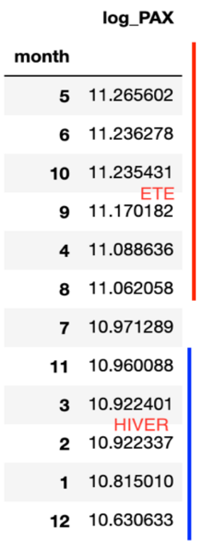
The aim of this report is to narrate our workflow, to detail everything we tried and explain the choices we made to build our best model and predict the number of air passengers.

I. Starting point, descriptive statistics and data preprocessing

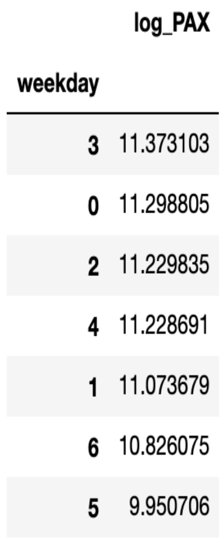
Understanding the task and how to proceed was our first challenge, as it was for both of us our first data science competition. Our starting point was to look at the data we were given. We tried to find patterns and correlations between variables, to detect missing values and to see which variables could help us predict best the number of air passengers.

A. *'Date'*

We started by looking closely at the date after one-hot encoding it. Our analysis showed us that there was a difference in the number of air passengers for the different weekdays and the different months. Therefore, we built a variable '*weekend*' and a variable '*season*' to better capture this information. Creating these variables improved our score when we used specific regressors such as SVM. However, they didn't improve our score in our gradient boosting model, so we didn't keep them in our final submission.



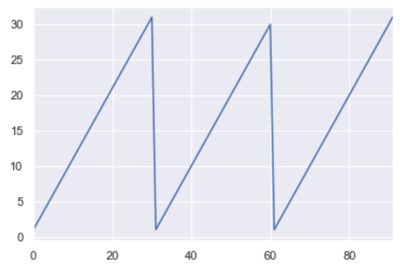
month	log_PAX
5	11.265602
6	11.236278
10	11.235431
9	11.170182
4	11.088636
8	11.062058
7	10.971289
11	10.960088
3	10.922401
2	10.922337
1	10.815010
12	10.630633

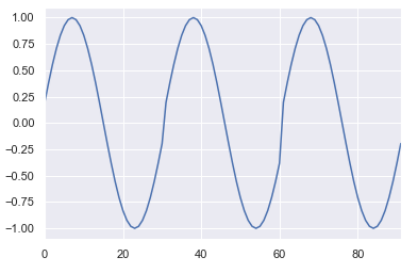


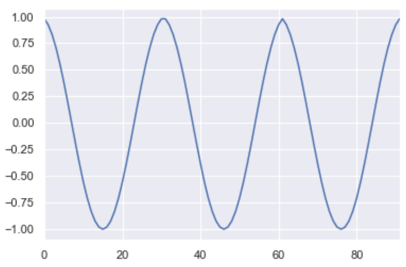
weekday	log_PAX
3	11.373103
0	11.298805
2	11.229835
4	11.228691
1	11.073679
6	10.826075
5	9.950706

We also noticed that the variables which represent the date ('*month*', '*day*'…) weren't continuous. For the variable '*month*', December corresponds to 12 and January to 1, so the algorithm doesn't know that January follows December. Therefore, we designed a model in which we transform the different variables ('*day*', '*month*', '*week*', and '*weekday*') into two dimensions using cosines and sinus functions to represent the data continuously and this allowed us to improve our score. We also kept the column for the year of the flight (we subtracted 2011 to the 'year' variable to have an encoding that goes from 0 to 2) and the day number of a particular date starting from the first date (2011-09-01) of the dataset.

The first graph shows that if we encode days with their number in a month, every 30 or 31 days, the graph isn't continuous. However, when we use our transformation (second and third plots), the values go from -1 to 1 and are always continuous.







B. *'Airports'*

Then, we concentrated our work on how to encode the departure and arrival airports. The first method was to use a one-hot encoding. However, this technique leads to 40 columns (two columns per airport) and we figured that we didn't need that many. Our idea was to create a column per airport and put the value 1 if the airplane leaves this airport and -1 if it arrives there. The number of columns was divided by 2. Lastly, we decided to implement a mean encoding method (we multiplied the airport columns by the average number of passengers respectively leaving or arriving in that airport). This encoding allowed to improve the score of our final model.

C. *'Reservations'*

Finally, we focused on the reservation variables ('*WeeksToDeparture*' and '*std_wtd*'). We found a strong correlation between these two and tried different methods to handle the information contained in the two columns. We tried to replace the two variables by a single product column of the two. We also tried to use a PCA algorithm with one principal component, but in the end, we decided to simply drop the variable '*std_wtd*' and to create different categories for the variable '*WeeksToDeparture*'. We created four groups of equal sizes (using *pd.cut*) by ordering the variable '*WeeksToDeparture*', and we encoded these groups from 0 to 3.

II. Adding and preprocessing external data

A second step was to look at the external weather dataset. We analyzed the correlation between the weather variables and the number of air passengers and tried to:

- regroup variables using a PCA algorithm with a smaller number of principal components,
- delete some redundant information (minimum and maximum of a variable),
- do operations between them (like multiplying humidity and temperature).

We ended up only keeping the variable '*Events*' which we thought could have an influence on the number of passengers. As this variable was categorical, we first replaced missing values by 0 and we encoded into three numerical values: 0 when there were no particular events, 1 for common events such as rain or fog, 2 for more important events such as thunderstorms.

Overall, we thought that the weather data wasn't necessarily the best thing to improve our model. In fact, the weather on the day of arrival will not necessarily determine the weather of the following days for a week of holiday for example.

As the external data we initially had didn't allow us to explain enough the number of air passengers, we decided to add new external data.
Overall, we added the following four variables:

- Distance in kilometer between the airports: '*Distance*'
- Number of inhabitants per city: '*Population*'
- Revenue per state of the arrival airport: '*Revenue*'
- Crude oil price: '*Oil_Price*'

Every time we added a new variable, our technique was to also challenge the other variables. We didn't want to keep information if they weren't relevant. Therefore, each time we added new data to our model, we tested to remove others to see if they still had an impact on our model or if the accuracy was still the same without them.

A. *'Distance'*

Our first idea was to add the distance between the airports. We had the intuition that longer flights would carry more passenger and we thought that the distance information could help our algorithm to make a difference between long flights with more passengers and shorter flights. We found the geographical longitude and latitude of each airport and used a function to calculate the distance between each of the twenty airports. This information consequently improved our score, so we decided to keep this variable. We tried at some point to regroup the distance between airports into three categories of flights that we could encode: long flights, medium flights and short flights. However, this encoding did not allow us to improve or score and we lost a bit of precision, so we decided not to keep this encoding.

B. *'Population'*

Our second idea was that the number of passengers on a flight could be explained by the population of the city of arrival and that flights going to big cities would carry more passengers. Therefore, we created a column with the number of inhabitants in the city of arrival. We kept it during a long period of our tests, but in the end, we realized that it didn't explain significantly our response variable and didn't keep it in our final model.

C. *'Revenue'*

Related to the number of inhabitants of a city, we also thought that the per Capita income could represent the resources of a state. The richer a state is, the more its infrastructures are developed, and the more travelers and workers are coming and going. This variable slightly improved our score, so we decided to keep it after having challenged it with the other variables. We also tried to use it differently and we created 3 categories of revenues. We affected 0 to the airports of a low-income state, 1 to the airports of a standard income state and 2 to airports of richer states. However, this try wasn't effective and we kept the numerical values.

D. *'Oil_price'*

Lastly, looking at the bigger picture, we focused on the expenses of an airline company. Most of them are related to the airplanes and their maintenance, the crews, and oil. Since this last expense has a cost that varies enormously, its variations may be reflected in air tickets' prices. So, we added this variable to our external dataset and tried different techniques to use it. When we first added this variable, it wasn't relevant. The oil price of the day of the flight isn't influential on the tickets' prices since they are established a long time before. So, we tried another approach and we associated with a flight the oil price three months before it. After adjusting several times this number of months, the variable still wasn't relevant. So, we decided to drop it too.

III. Choosing a machine learning algorithm

After comparing the results from several regressors among which Random forest, SVM and Logistic regression, we decided to go for a gradient boosting algorithm. More particularly, we opted for the *XGBRegressor* as it fitted the best our data and gave us our best score.

Some advantages of this algorithm are:

- Speed: *XGBRegressor* is quick to execute thanks to parallel processing.
- Performance: *XGBRegressor* outperformed the other algorithms we tried.
- Regularization: *XGBRegressor* has in-built L1 (Lasso Regression) and L2 (Ridge Regression) regularization parameters that we can adjust to prevent the model from overfitting.
- Not too dependent on the number of variables.

We wanted to find another regressor that would perform as well but goes faster. We tried to use the *LGBMRegressor*, which is another gradient boosting regressor. This new algorithm was performant and a lot faster when we used it with the local test (RAMP_submission_test). But, when we submitted our work using this regressor, we got good results but not as good as with the *XGBRegressor*, and the time wasn't reduced a lot (only 0.5 seconds faster).

Finally, we found the techniques of model blending and stacking.

- Blending consists of combining information from multiple predictive models to generate a new one. By trying the method, we used different machine learning algorithms and took the mean (with desired weights) of the predictions we obtained using these different models. We tried this method with our two best algorithms (*XGBRegressor* and *LGBMRegressor*) and our score was improved.
- Stacking is a more complex technique. After combining information from multiple predictive models, you use another model (called the stacked model) that will perform a prediction using the predicted values of the base models. It highlights each model that performs best and discredits the ones that perform poorly. Therefore, the technique is most effective when the models are significantly different.

These two methods produce small gains and add complexity to your model. Therefore, one of the main disadvantages of these methods is the increase in running time. Depending on the context, these techniques may be useless and inefficient.

In our case, the first technique helped us to improve our score by one percent.
We also tried the stacking technique with the *LGBMRegressor* and *XGBRegressor* as base models, since we had our best scores with them. Even if they are quite similar (two gradient boosting algorithm), their predictions are slightly different in our case (*XGBRegressor* overfits more than *LGBMRegressor*). As stacked model, we chose the *KernelRidge* algorithm with a polynomial kernel. Since the task of the stacked model is to fine-tune the prediction of our two base models, we wanted to find an algorithm that runs fast. We tried to use a *LinearRegression*, but a polynomial seemed to be more adapted, and *KernelRidge* is faster than the *SupportVectorMachine* algorithm for the size of our dataset.
However, this technique didn't increase our performance, it only increased the complexity of our model and our running time, so we decided to keep our model with the blending technique.

IV. Parameters tuning

Every time we tried a new model or tried an existing model on new data, we tuned and adapted the parameters of our regressor to fit the data as well as possible. To do so, we used a cross-validation *GridSearch* based on minimizing the RMSE.

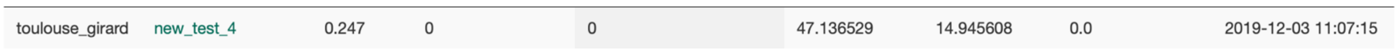
We will focus on the parameter of the *XGBRegressor* since we spent a lot of time tuning his parameters and it is at the center of our model. Therefore, the most important parameters are:

- *max_depth = 7:* Maximum depth of a tree. We Increased this value to make the model more complex and even though it is more likely to overfit we got better results.
- *n_estimator = 2450*: number of boosted trees. We set a high value to improve our model.
- *learning_rate = 0.16:* learning rate of our model.
- *min_child_weight = 5:* Minimum sum of instance weight needed in a child. We increased it too to make our model more conservative.

Overall, our parameters tend to make our model overfit the train data (we noticed a quite important difference in our results between the train and the test set) but these are the parameters that gave us the best score for the competition.

V. Score obtained, lessons learnt and conclusion

Finally, our model obtained a RMSE of 0.247.



toulouse_girard	new_test_4	0.247	0	0	47.136529	14.945608	0.0	2019-12-03 11:07:15

Our score of 0.247 was a combination of all the things we tried during these several weeks of competition: adding new useful variables to our dataset, preprocessing all our variables, choosing the best machine learning algorithm and tuning its parameters to fit our data.

Our workflow was very circular: every time we added new data or did relevant feature preprocessing, we would challenge other variables in the actual model, and analyze how our score was impacted. We also tried to choose the best regressor and adapt it each time so that it would fit the dataset. We tried to improve our score at every step of our workflow and would continuously go back and forth all the steps.

Overall, this project was particularly interesting to understand the importance of data preparation and the particularities of the different regressors. It was also fun to look for new external data and see how they could improve our model. Undoubtedly our model could still be improved by adding more data and going deeper into parameter tuning.