RAMP Challenge Final Report

Predicting Air Passengers

Jiayu Gan, Honghao Yu

I. External Data

Predicting air passengers can be a challenging task when the original data contained only limited information about the departure and destination airports and ticket booking time. Many factors might have accounted for the actual number of passengers present on a specific airplane on a specific date, like the economic vibrancy of the two cities, the weather, the airport operating conditions, historical on-time rates, number of flights connecting the two cities, competition among airlines, a vacation or a special event, etc. We were lucky to have the weather data to begin. Apart from that, we have been able to find detailed information about airport operations (departures/arrivals, passenger volume, etc.) and flights (on-time/delay, frequency, etc.) of each day on the Federal Aviation Administration (FAA)'s website, and general info about airports (locations) and routes (competing airlines, distances) on OpenFlights. Besides, we have included data about the socioeconomic conditions (GDP, population, etc.) of the cities obtained from Wikipedia and the Federal Reserve Bank (FRED). Considering the seasonal travel pattern, we have also included the major public and school holidays in the US. For a comprehensive list of the external data we've used and their sources, please refer to **Appendix A**.

II. Preliminary Exploration and Model Selection

For the first trial, we used monthly data on air traffic and transformed the departure and destination airports into a series of 0-1 dummy variables (40 dummies for each observation). We considered indicators of extreme weather which might lead to flight delays or cancellations like minimum temperature, maximum wind, visibility, etc. and explicitly included dummies for thunderstorms and snowstorms. Another dummy was used to suggest whether the given day was a public holiday.

Unfortunately, after experimenting with MLP, SVR, RF, etc. from Scikit-learn, our best results (RMSE) remained above 0.4. We then tried Gradient Boosting Decision Tree (GBDT) and successfully lowered the RMSE to 0.37. This was the starting point for our trials with ensemble methods, where multiple weak learners integrate into a much stronger learner. The base learners yield inaccurate predictions individually, but if ensembled in a wise way, they can integrate into a powerful learner by learning higher-order interactions between features. Specifically, in Gradient Boosting, the aim of each base learner (a decision tree) is to fit the residual of the preceding base learners, or in other words, the newly-added base learners can move the model a step further in the direction of the negative gradient of the loss function and thus the overall RMSE can be reduced.

Thanks to the feature importance report provided by GBDT, we figured out that most weather metrics were poor predictors of air passengers and it would be a bad idea to create a dummy for each airport. Based on the satisfactory results of GBDT, we explored other ensemble methods like AdaBoost, LightGBM and XGBoost. Among these, XGBoost gave more accurate predictions at a faster speed. XGBoost model is an extension of Gradient Boosting in the sense that it utilizes information from both the first-order derivative (gradient) and the second-order derivative (Hessian) of the newly-added tree. Furthermore, XGBoost provides Scikit-learn style API, allowing us to use auxiliary modules in Scikit-learn, such as *cross-validation*, *RFECV*, *grid search* and *random search*. Therefore, we decided on using XGBoost as our primary model.

III. Feature Transformation and Preprocessing

The preliminary explorations gave us a clearer intuition of the features that might have weight in this case. As the monthly data were too rough to make good predictions, we replaced them with daily data. We then dropped all the weather features except for daily average temperatures and maximum wind. In terms of airports, we used their geographical coordinates (longitude, latitude, altitude) and socioeconomic conditions (GDP, population) of the cities they serve to replace the dummies. To capture the pairwise effects, we added the number of flights, the number of operating airlines, the route mileage for each given pair of departure and arrival airports, all on a daily basis.

It's reasonable to assume that negative prospects for on-time travels would dampen passengers' interest in taking a flight, so we added a whole bunch of metrics for flight schedule reliability (on-time rates, average delay, cancellations, etc.). We found them jointly significantly but not so much individually, thus we performed a PCA to extract the most valuable information. The first three components accounted for 95% of the total variance and were therefore selected.

Another reasonable assumption is the load factor (total onboard passengers / total seats) of a particular airplane on a given date would be to some extent in accordance with the overall load factors of the departure or destination airports on that day. For this reason, we calculated the daily average departure and arrival seats and load scores of all the 100-199 seat airplanes for each airport.

The seasonal pattern has been a prominent feature of the transportation industry, so we created a variable *week* to locate a day within a year and 7 dummies (Mon, Tue, Wed, etc.) to locate it within a week. Furthermore, we included the days before and after a major public or school holiday with an additional categorical variable describing the length of the holiday.

We also included the change in latitude and longitude from the departure airport to the arrival airport to indicate the direction of flight. In order to capture some particular patterns (e.g. many Americans from the north would like to travel to south for warm sunshine in winter), we introduced dummy variables indicating the interaction between flight direction and season.

After merging the external dataset, we found 4 missing entries in the whole training set. Imputation with the median or mean value of the corresponding features is a reasonable and frequently-used approach. But we found that replacing them by 0 (before preprocessing) yielded a better score. The base learner of XGBoost is a decision tree, which generally does not rely on feature standardization as much as other regressors such as linear regression and neural networks. However, we still observed a drop in RMSE after we standardized the features into Z-scores. To reduce the impact of outliers, we replaced the Z-scores outside the interval [-3, 3] by -3 or 3 depending on their signs.

IV. Feature Selection

In total, we had around 200 features, either collected or derived from existing ones. Compared to other machine learning models, XGBoost is more robust to redundant features. However, we observed that the accuracy of prediction would drop slightly if we feed features without filtering (around 10% higher RMSE than optimal). Besides the feature importance scores provided by XGBoost, we can get more helpful information from the RFECV module provided by Scikit-learn. RFECV is able to conduct backward feature selection without specifying the number to be selected. However, the result of RFECV is not necessarily optimal, since it iterates through backward selection path only. We used the list of selected features provided by RFECV as a reference and then conducted both forward and backward stepwise feature selection manually. As described in **Section V**, the model used to select features had already been fine-tuned, so the features selected were considered to be final. We selected 19 features among 200 to feed the model. These included time metrics (weekday, week, month, year, days before/after a major holiday, etc.), air travel market metrics (number of flights, number of competing airlines, etc.), socioeconomic metrics (GDP, population) and geographic attributes (longitude, latitude, altitude, distance). For a comprehensive list of the selected features and their details, please refer to **Appendix B**.

V. Parameter Tuning

Thanks to the powerful XGBoost algorithm we got a reasonable model with default parameters. It generally outperformed our fine-tuned GBDT model, with higher accuracy, less time consumption, and less overfitting. The XGBoost model with default parameters set a good starting point, allowing us to go further by fine-tuning the hyper-parameters.

We first looked for a reasonable *n_estimator* associated with the *learning_rate* with the help of an early-stop technique. From the training/validation loss curve, we knew that the XGBoost model is robust to overfitting, which means we can lower the *learning_rate* without incurring a large loss but only to increase the time consumption. At this stage, we chose a relatively large *learning_rate* of 0.05 and a corresponding *n_estimator* of 2000 to be time-efficient during the tuning. The early-stop module told us that the validation score remained constant after 2000 iterations.

After fixing the *n_estimator* and *learning_rate*, we focused on the parameters that control the shape of decision trees, *max_depth* and *min_child_weight*. The former limits the depth of decision trees and the latter prevents an arbitrary node with few samples from being further split. We used the grid search method to get the optimal combination: *max_depth* of 7 and *min_child_weight* of 6. By doing this we managed to reduce the validation RMSE to 0.33, which is far lower than what a fine-tuned GBDT model could achieve. The model was still somewhat overfitted with a training RMSE of 0.07, leaving us the potential to improve by dealing with overfitting.

XGBoost offers a comprehensive diagnostic toolkit to adjust the model, such as *subsample*, *colsample_bytree, colsample_bylevel,* and *L1/L2 regularization.* By modifying these parameters we can make the model as conservative as we want, depending on the balance between bias and variance we want to achieve (in general, a more conservative model has a larger bias but a smaller variance).

Since the parameter space we need to search into was much bigger than the one consisting of *max_depth* and *min_child_weight* (5 continuous variables versus 2 integers), we used a random search with 5-fold cross-validation in a wide range and then a grid search in the narrowed range near the sub-optimal combination we've got from random search. Theoretically, if we modify the subset of features fed to the model, the optimal hyper-parameters will change accordingly. We need to iterate back and forth between parameter tuning and feature selection, which is computationally intensive. But in practice, we found that the optimal combination of parameters was rather robust as we changed the subset of features. We thus saved time by skipping redundant iterations. The fine-tuned combination of hyperparameters is listed in table 1. All unmentioned hyperparameters were remained as default.

Table 1. Fine-tuned combination of hyperparameters

Category	**Hyperparameter**
Model Complexity	*max_depth:* 7, *min_child_weight:* 6, *n_estimators*: 8000
Subsampling	*subsample: 1, colsample_bytree:* 0.36*, colsample_bylevel:* 0.74
Regularization	*reg_alpha*: 0.001, *reg_lambda*: 0.001
Training	*learning_rate*: 0.01

VI. Result Interpretation

Most machine learning techniques generally work as magical “black boxes”, making sense only in the global level, and have long been criticized for their poor interpretability of individual features and instances. However, the newly-proposed SHAP (SHapley Additive exPlanation) value enabled us to dive down to the individual level and give interpretations that are more meaningful. The Python *shap* package provided a rich set of tools to visualize the impact of the features on the predicted values and here we’ll briefly talk about the mean value bar plot (Figure 1), the scattered summary plot (Figure 2) and the force plot for an individual observation (Figure 3), which could be found in **Appendix B**.

The first plot showed clearly that the most powerful predictor was the number of flights from the departure airport to the arrival one on the given date. This was followed by weekday and wd_5 (whether it was a Saturday). Overall the time metrics as a whole proved to be strong predictors and the socioeconomic metrics worked fine as well. Geographical features while useful, didn’t change the predictions as much, with the exception of distance. A close look at the scattered plot revealed that a larger volume of passengers can be attributed to a greater number of flights, a higher GDP of the departure and arrival city and a longer distance between the two cities, which made sense intuitively. The days to the next and last holiday worked in the opposite directions as we had expected. The closer to the next holiday, the larger volume of air passengers. What we hadn’t expected though was that altitude played a role and that a lower altitude was favored by passengers. We also made a force plot for a specific observation. In this case, as illustrated by Figure 3, compared to the baseline value (mean value of all the predictions) 11.0, Flight_Count being +0.8 std from mean shifted logPAX up by around 0.25 (roughly equal to a 25% increase in PAX), Gdp_y_log being -0.9 std from mean shifted logPAX down by around 0.15 while Gdp_x_log being -1.1 std from mean had approximately the same downside effect. These being the major drivers of this particular instance did not contradict with the mean SHAP values, as it is exactly SHAP’s merit to allow for individual aberrance.

To sum up, the external data and the model we’ve chosen had successfully brought down the RMSE with 10-fold cross-validation to 0.30±0.03 locally. Considering the easy availability of the external metrics and the robustness of XGBoost, our model would seem like a good option for future predictions.

VII. Conclusion

In this interesting project we were presented with the challenge to predict air passengers on the basis of a thin existing dataset. After iterating back and forth between choosing then fine-tuning different models from Scikit-learn and experimenting with different sets of features obtained from various sources, we found decision-tree-based XGBoost to be a highly efficient and robust model with significantly lower RMSEs than others could achieve. We then focused on utilising random search and grid search to obtain the optimal set of hyperparameters. We were lucky to find that the optimal hyperparameters seemed insensitive to the change of feature set and thus avoided exhaustive search. With the hyperparameters fixed, we kept navigating with the features either by trying different combinations or by doing transformations. A pool of around 200 features either collected or derived had been screened by forward and backward selection techniques. We finally ended up with 19 features which altogether brought about the strongest predicting power. Among these were air travel market metrics, time metrics, socioeconomic metrics and geographical attributes. These features were mainly official statistics and thus very unlikely to produce aberrant values which might result in prediction failures. With the help of SHAP value framework, we figured out the relative importance of these 19 features and how each observation was impacted by them. The number of flights on that day, which weekday it was and whether it was a Saturday seemed to be the best predictors of the volume of air passengers from one airport to another. Further deep-dive analysis is to be done to reveal the hidden effects of the features and their mutual interactions.

Appendix A:

Table 2. Comprehensive list of external datasets and their sources

Dataset	**Source**
On-time & Delay statistics by airport	FAA: https://aspm.faa.gov/asqp/sys/Airport.asp FAA: https://aspm.faa.gov/asqp/sys/Airport.asp
Airport Geographic Coordinates	OpenFlights: https://openflights.org/data.html#airport
Details about Flights between the Two Cities by Day	FAA: https://aspm.faa.gov/apm/sys/AnalysisCP.asp
Route Distances	Derived from airports' geographic coordinates (using geopy.distance.distance() function based on Haversine formula)
Departure & Arrival Seats by Airport by Day	FAA: https://aspm.faa.gov/tfms/sys/Airport.asp
Socioeconomic Conditions of Cities Served	Wikipedia (keyword: city names); FRED: https://fred.stlouisfed.org/series
School Holidays	https://www.feiertagskalender.ch/ferien.php?geo=3537&jahr=2012&klasse=0&hl=en
Public Holidays	Holiday Package PyPI

Appendix B:

Table 3. Comprehensive list of selected features and their descriptions
(in descending order of SHAP values)

Feature	**Description**
Flight_Count	integer; number of flights between the pair of airports (uni-direction) on the given date
weekday	ordinal; 0 - Mon, 1 - Tue, 2 - Wed, 3 - Thu, 4 - Fri, 5 - Sat, 6 - Sun
wd_5	dummy; 1 - is Saturday, 0 - is not Saturday
Gdp_x_log	float; logarithmic form of the GDP (in 2017) in $B of the metropolitan area served by the departure airport
week	ordinal; ordinal number of the week within a year that the given date was in
distance_KM	float; distance between the pair of airports measured in kilometers
Gdp_y_log	float; logarithmic form of the GDP (in 2017) in $B of the metropolitan area served by the arrival airport
n_days	integer; number of days beyond 01/01/2011
DaysToProchaine	integer; days to go until the next holiday (public & school)
Airline	integer; number of competing airlines operating the route between the pair of airports on the given date
Pop_x_log	float; logarithmic form of the population (in 2018) of the city served by the departure airport
DaysToDerniere	integer; days beyond the last holiday (public & school)
Pop_y_log	float; logarithmic form of the population (in 2018) of the city served by the departure airport
Lat_y	decimal(6, 3); latitude of the arrival airport
Lon_y	decimal(6, 3); longitude of the arrival airport
Lat_x	decimal(6, 3); latitude of the departure airport
Lon_x	decimal(6, 3); longitude of the departure airport
Alt_x	float; altitude of the departure airport in meters
month	ordinal; month

Figure 1. Mean SHAP values of the features

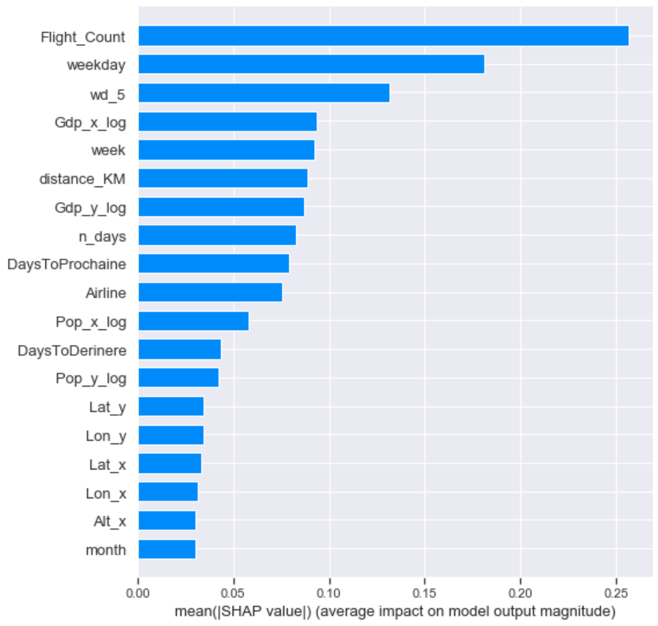


Figure 2. SHAP values of the features at the individual level

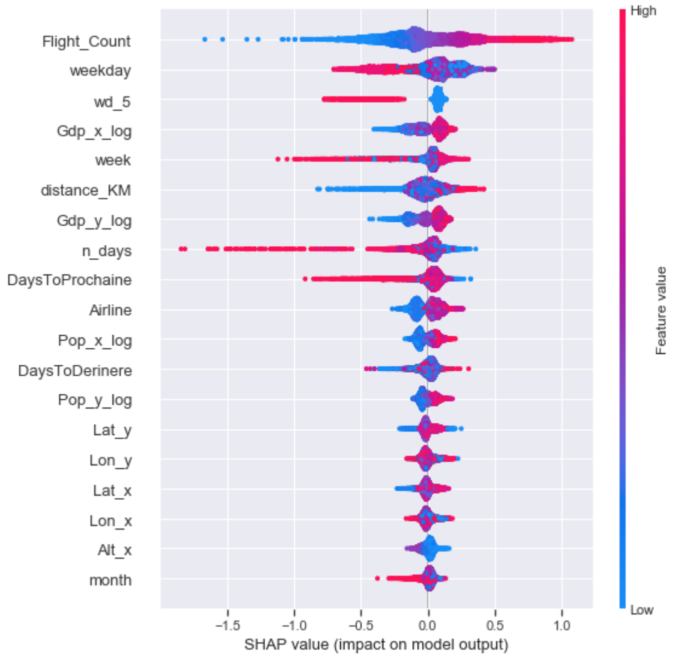
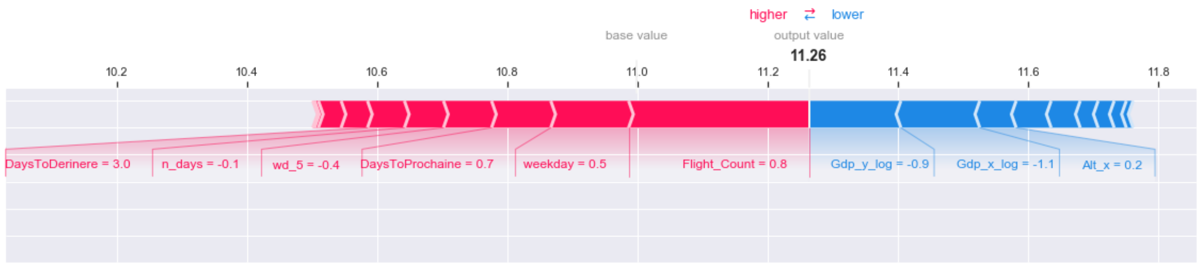


Figure 3. Force plot for observation #6530


higher ⇄ lower
base value
output value
DaysToDerinere = 3.0
n_days = -0.1
wd_5 = -0.4
DaysToProchaine = 0.7
weekday = 0.5
Flight_Count = 0.8
Gdp_y_log = -0.9
Gdp_x_log = -1.1
Alt_x = 0.2