Airplane Passenger Load Prediction

Tommy TRAN, Thomas de MAREUIL - RAMP Submission: *2Tomsbetterthan1*

1. Context and objective

This report is our final project for the class *MAP536 - Python for Data Science*.

Our objective was to **forecast airplane passengers load** in the United States. We were initially provided with:

- training data : information about US domestic flights between 2011 and 2013
- external data : meteorological information about US airports

Based on these first pieces of information, we identified several steps to work on and improve our predictions:

1. **Feature engineering**:
   → look for additional external data
   → clean it and merge it with preexisting data
   → adjust data encoding
   → check features relevance

2. **Model selection and tuning:**
   → try several models and select the best-performing
   → try model averaging and stacking
   → tune model hyperparameters with gridsearch

In this report, we will go through these steps and present you the ideas we got, the techniques we tried, the models we explored and the results we obtained.

2. Data Exploration and Feature Engineering

A. Structure of the initial datasets

The initial **training dataset** contained information about 8902 US domestic flights: date (from 09/01/2011 to 03/05/2013), departure and arrival airports (20 different airports for both), how early tickets were booked (in weeks) and the number of passengers (our predicted variable).

→ This dataset contained no missing values and low skewness. Based on correlation plots we observed that `std_wtd` (standard deviation of weeks to departure) is strongly correlated to `WeeksToDeparture` and little correlated to `passengers`, therefore we will not to use it in the regression.

The initial **external dataset** contained information about meteorological features for the 20 airports, during the corresponding time period. Each line represents the property of an airport at a given date.

→ Based on correlation, we kept only the Mean values for each feature that also had Min and Max values (i.e. `Temperature`, `Humidity`, `Wind`, etc.).

B. Adding more external data

We tried to identify the determinants of airplane passenger load, and after online research we came out with:

General / time-related data:

- **Holidays**: we added a binary indicator to identify holidays (ex: Matin Luther King Day, July 4th, Thanksgiving, etc.), with more plane traffic.
- **Plane load factor**: percentage of empty seats in US domestic flights, obtained from the US Bureau of Transportation Services (BTS) database. We only found the average value per month, over the whole US market, which isn't very precise but gives and idea of plane load tendencies over the observed time period.
- **Jet Fuel Prices**: found in the IATA (International Air Transport Association) database, jet fuel prices tend to impact the price of tickets and therefore passenger load.

Data per airport:

- **Airports ranking** and **frequentation** in terms of commercial passengers per year (IATA).
- **City population and density** (IATA).
- **Latitude and longitude (distances)**: the IATA database gave us latitude and longitude coordinates of the airports. We then explored the pyproj and geopy packages to compute distances, but we finally chose to use direct computation with the Haversine distance formula, which we embedded in the feature extractor.

Data per flight route:

- **Air time** (BTS): time spent in flight to link 2 airports.
- **Average ticket prices** (US BTS): average ticket prices between 2 given US airports, which has a direct impact on passenger load. We did not find day by day values, but we obtained an averafe price per quarter over the observed period, for each of the relevant flight routes.
- **Average daily number of passengers** for a given flight route (BTS): similarly, we found a quarter-by-quarter value of the daily number of passengers on the relevant flight routes, during the observed period.

C. Data pre-processing: cleaning, merging, encoding

Producing a clean external dataset

A tricky part was first to clean the data we found online, and then to merge it into a single `external_data` file, including new and pre-existing external data (as we needed to have a single file to upload on RAMP).

To clean it and format it, we used mainly `pandas`, `numpy` and `datetime`. We had, for example, to drop and select values, to perform computations on columns, to match city names with airport codes, to change data types, to manipulate dates and times, to reindex rows, etc. As we found extensive data online, we were not confronted with the problem of missing values.

In the end we produced 4 external datasets to be merged together into a single file: the original weather data (sorted out by airport and date), our new general data (holidays, fuel prices...), our new data per airport (airport ranking, latitude, longitude, city population...) and our new data per flight route (flight time, average daily number of passengers...).

Weather data, airport data and general data were easy to merge together on the `airport` and `date` columns (which they had in common), using the `pd.merge` function. Merging was more complicated for the fligh route data, as information *didn't correspond to a single airport*. As we could not keep 2 separated files, we just **concatenated** our data per flight route next to the rest of the data (using the `pd.concat` function), to produce a final `external_data` file.

Merging with the training dataset

We built the feature extractor in order to select features from our final `external_data` and merge them with the training data.

The training data is sorted out by flight route, so part of our external data (the flight route data) was easy to merge. We just had to add a flight route column in the training data, select the external data related to flight routes, and perform the merge on the new route column. We also split the dates in the training dataset using `datetime` in order to have months/quarters appear and to merge on the correct dates.

For the other features, related to single airports, we had to merge them *twice* with training data, based on the 2 airports in the itinerary (once on the Departure airport, once on the Arrival airport). Our feature extractor adds `_dep` and `_arr` suffixes in the column names in order to distinguish information related to departure and arrival airports. Once the merger done, we used the Haversine formula to compute distances between latitude and longitude coordinates of departure and arrival airports.

Encoding categorical features

Before feeding the model, we still had to encode the categorical features, i.e. dates (split into year, month, week, day, weekday) and airports (Dep and Arr). We tried several encoding methods, especially **target encoding** and **one-hot encoding**. Target encoding did not yield efficient results, therefore we chose to stick to one-hot encoding.

Drop irrelevant features

We made several *back-and-forths* with the model to test differents sets of features and assess them with **feature importance analysis**. We then included a line in the feature extractor to drop the least significant features, and produce our final `X_encoded` file, to be fed to the model:

In [10]:
```
print(X_encoded.shape)
X_encoded.head()
```

```
(8902, 119)
```

Out[10]:

	WeeksToDeparture	Mean TemperatureC	Rank_2018_Dep	population_Dep	density_Dep	Fuel_p
0	12.875000	29	3	8675982.0	4612.0	
1	14.285714	25	9	2073045.0	1747.0	
2	10.863636	19	5	2787266.0	1774.0	
3	11.480000	19	1	5228750.0	1384.0	
4	11.450000	12	5	2787266.0	1774.0	

3. Prediction

A. Model Selection

We tried and evaluated several regression models locally, in order to obtain the lowest error rate on our testing set. Here are the results of our different tries, ranked from lowest to highest test rmse:

- **Exterme Gradient Boosting (`XGBoost`)**: the best-performing model. This is **the model that we finally selected**.

```
----------------------------
train rmse = 0.137 ± 0.0025
test rmse = 0.267 ± 0.0072
```

- **Gradient Boosting**: 2nd best performer, just a litle behind `XGBoost` .
- **Random Forest**: the initial model in the starting kit, outperformed by gradient-based models.
- **Lasso Regression**: this method and the subsequent ones were significantly less performant.
- **Elastic Net Regression**
- **AdaBoost**
- **Linear Regression**

Model averaging / stacking

We tried **averaging** the 3 best-performing models (XGBoost, GradientBoosting and RandomForest), which did not yield better results - probably due to the fact that these models are already ensembling methods.

Overfitting

In the XGBoost case the model seems to overfit a little bit, but not significantly.

B. Hyperparameters tuning

We performed Grid Search using the `GridSearchCV` command (the `RandomizedSearchCV` is too costly), which is still quite demanding computationaly speaking but allowed us to come up with an efficient set of parameters for our XGBoost, gaining **-0,17** in test score compared to default parameters. Below, our final model:

In [ ]:
```
XGB = xgb.XGBRegressor(colsample_bytree = 0.7,
                       learning_rate = 0.05, max_depth = 10,
                       min_child_weight = 4, n_estimators = 5000,
                       nthread = 4, objective = 'reg:linear',
                       silent = 1, subsample = 0.7)
```

4. Conclusion

Answer to the problem

Using the model we built, an airline company would be able to predict passenger load on a given route and day. This could help them **allocate planes** (e.g. schedule more flights on highly-frequented routes), or tune their **pricing policy** (increase ticket prices on highly-frequented routes). Indirectly, it could also help them adjust their **resource management**, by allocating more resources (investments, additional staff, etc.) to the routes with the highest passenger load.

In addition, the predictions could be **further analysed** (for example grouped by week or month, comparde with other airline companies, etc.) in oder to produce additional insights about **what determines passenger traffic the most**, what are the **preferred dates**, the **preferred routes**... These additional analyses would very provide valuable information to the airline company in order to **help them improve business performance**.

What we learnt in this project

We confronted ourselves to the different steps of a data science project, from obtaining and pre-processing the data to choosing and tuning the model. We tried many different data manipulation techniques and explored several models along the way, with a constant back and forth between model tuning and features engineeering. We also learnt to use github in a team project.

→ This experience allowed us to apply what we learnt in class, and most of all it helped us learn a lot by exploring the internet (stack overflow, etc.) to find solutions to our problems, in a "learning by doing" approach!