Airbnb price prediction in Paris - Technical documentation

Péter ENDES-NAGY

Introduction

This is a technical documentation of the *Report on Parisian Airbnb price prediction* (available on GitHub).

Data

Data was downloaded from the official Airbnb website (link here) for the city of Paris, France, date of 07 June, 2021. The codes (available in this GitHub repo) are suitable to run on other dates, but might need tailoring for other cities.

Raw data preparation

Code used for this part: available here) The raw data was heavily transformed before the analysis-relevant feature engineering. The original raw dataset contained empty/unnecessary/redundant variables, datatypes had to be fixed, and new variables created/transformed.

1. **Dropping** unnecessary and empty **variables**. Some kept for cross-checks, dropped later on.

2. **Dealing with amenities**. They are stored in a single column, in a not data-frame friendly format.

A) All possible amenity types were retrieved, cleaned and merged, duplicates removed, ended up with 1010 amenity types in total. Only those were kept that have at least 1000 occurrences in the whole dataset: 73 differently worded amenity type.
B) The amenity types might be overlapping (information on parking with different wordings, etc.), but at the moment we focus on different wordings.
C) 73 binary variables were constructed, taking value of `TRUE` if the given amenity type is marked by the particular listing, otherwise `FALSE`.

3. **Fixing data types** Some numerical variables are stored as text, NA values weren't recognized by R.

A) NA values stored as `N/A` in character variables. Replaced with missing value.
B) Binary variables stored as text (`"t"` and `"f"` , missing value as `""`), transformed into `1 , 0 , NA`. The amenities stored as boolean also transformed into `1 , 0`.
C) Percentage variables stored as text. `%` marks removed and transformed into numerical values.
D) Number of bathrooms variable empty in the dataset, the relevant information is stored in `bathrooms_text`. Number value retrieved from the text variable. In some cases, no numerical value in the text, value 1 taken. Where length of the text is 0 (`""`), missing value introduced.
E) Fix price variable. Values stored in a text, currency and thousand separators included. The code works with US dollar signs only, other cities might call for modifying the code. In case of multiple currencies, further transformation would be necessary.

F) Later turned out that the fnóunky French characters are screwing up the csv reader, so they were removed from the text (replaced by ASCII characters).

The pre-cleaned and transformed dataset was uploaded to GitHub for later use and filtering. Available: here

Data cleaning and feature engineering

Code used for this part: available here) In this section, feature engineering steps were carried out. Dealt with missing and extreme values, decisions were made on pooling and transforming some variables. Business case relevant filtering and consistent renaming of the "final" variables was also done.

1. **Missing values**

A) There were no missing values for the target variable, `price`, but some took value of 0. Clearly a wrong data, their number is low (111), so these observations were dropped from the dataset.
B) Some variables were missing for a quarter of the observations. They aren't too relevant for the analysis so they were dropped. We also had many metrics on reviews, strongly correlated, so only the total score was kept for the analysis.
C) Data on the host was missing consistently across observations - same variables affected. The binary variables were imputed as 0 (not a superhost, doesn't have a profile picture, identity not verified), listings count imputed as 1, assuming at least 1 listing, as they are in the dataset. A flag variable was also introduced.
D) Missing data on number of bathrooms replaced with the median. A few observations marked 0 that feels like wrong data (maybe a shared bathroom wasn't considered) so it was changed to 1. Missing number of beds replaced by assuming double beds, so the number of accomodates variable was divided by two and rounded up. Missing data on number of bedrooms replaced by number of beds.
E) Missing review score data replaced by the median and flag variable created. number of reviews per month was missing where no reviews, therefore 0 values imputed. Date of the first review was replaced by `host_since` variable, if latter also missing, then with the `last_scraped` variable, assuming it is a fresh listing. This variable is used for calculating number of days elapsed, for how long the apartment has been on the market, so the bias introduced by the latter imputation is similar to the general bias.

2. **Create and transform variables**

A) The main question regarding the target variable: log or level (log is recommended for prices) and what to do with extreme values. *Extreme values* were identified with histograms and percentiles. Some are very extreme and unrealistic. The upper 0.1% of the sample starts from 2000 USD, similar to high-end luxury hotel prices in Paris, so reasonable for an entire luxury flat. Observations with extreme values above 2000 USD were dropped. The histogram also suggested a *log transformation*, as the distribution of prices follows lognormal distribution.
B) Number of days elapsed (since first listed) approximated with number of days elapsed between first review and date of scraped.
C) Hotel rooms (`room_type` variable) were dropped as they aren't relevant for our business case. The names were also shortened and simplified.
D) The `property_type` variable is inconsistent, contains room type information (e.g. Entire apartment, Single room in apartment) that was removed. Only the following (cleaned) types were kept. This part is very Paris-specific, for other cities the `property_type` variable might have a very different content and structure.

3. **Pool variables**

A) `property_type` pooled into two main types: `Apartment` and `House`.

B) `host_listings_count` simplified into 3 categories: `1 , 2-5 , above 5`. In some cases 0 that looks like wrong data, so replaced by 1, assuming at least one listing, since they are in the dataset with a listing.
C) Number of bathrooms simplified into `0 , 1 , above 1`. 0 is very rare, though.
D) `number_of_reviews` pooled into `0 , 1-50 , 50+`.
E) `minimum nights` pooled into `1 , 2-5 , 6-29 , 30+`. The `30+` observations are practically long-term offers, listed on Airbnb. We don't have information if our client would be also open to rent out their mid-size apartments to long-term tenants, so these observations aren't dropped.
F) Transform the "number of reviews in the last X" type of variables into binary.
G) The number of beds in itself might not be necessary as we already have an approximation on property size with the number of bedrooms variable. What rather matters is the ratio of beds and bedrooms. A `sep_bedroom` binary variable was created: takes the value of 1 is there are exactly 1 bed per bedrooms, so the bedrooms are separated. In case of 0, it's probably a couch or no bedroom at all, but staying in a living room, so the binary variable takes 0 value for these observations too.
H) Similarly, number of bathrooms relative to the number of beds was calculated and pooled into a binary variable called `one_bathroom_per_bed`. The number of bathrooms itself was also simplified into a dummy variable, `more_bathrooms`: one or more.
I) Availability during the next X days type of variables feel redundant to keep. Most people plan their trips 1-3 month ahead, so what might really matter is the availability in this time frame. 30 days availability was deducted from 90, so we can see how many days are available for the 1-3 months ahead planners. Based on density plots, there are 2 main type of properties: not booked at all and fully booked (or made unavailable by host, we can't really decide which of the 2). Binary variables were created for unavailable properties: considered as not avaialable under 5% (might be available for 1-2 days that usually conflicts with minimum nights), and for completely unbooked properties with availability above 95%.

4. **Filtering**

A) As our client is interested in renting out mid-sized apartments for 2-6 persons, observations with 1 or above 6 accommodates were dropped. There were some odd observations with 7-9 bedrooms and max 6 accommodates, discarded as well.

5. **Renaming**

A) variables were (re)named in a consistent manner so they can be easily identified later. Amenities start with `am_` , numerical variables with `n_` , categorical (not binary) variables with `fac_` , binaries with `d_` and flag binaries with `f_`.

6. For some numerical variables, **quadratic and cubic forms** were created to capture non-linearity in LASSO Models.

A) `n_days_since`: quadratic and cubic.
B) `n_accommodates`: quadratic and cubic. C): `n_bedrooms` (approximating apartment size): quadratic and cubic.

Model building

Code used for this part: available here)
1. Train and hould-out sets were created, both random samples from the whole dataset, former as 70% of the original cleaned dataset.
2. We are building 4 models: LASSO, CART, Random forest and another Random forest with auto-tuning.
3. Predictors grouped for use. A) For CART and Random Forest, all variables in the dataset are used and

let the machine decide which one is picked in which form. B) For LASSO, the quadratic and cubic forms are also included, as well as some interactions. The interactions were chosen based on field knowledge, how property type and bourough is interacting with some other variables: . 4. As the price follows lognormal distribution, log of the price was used in each models.
5. 5-fold cross-validation was defined.
6. Tuning parameters set for decent performance and running time: A) Minimum node size set to 50 so it won't take way too much time growing the trees and overfit. B) Split rule set as `"variance` C) `.mtry` parameter set to 10, number of variables randomly sampled for Random Forest 7. As autotuned Random forest (and other ML models) takes an eternity to run, the model results were saved into RData and called later.
Model evaluation Code used for this part: available here)

1. Tables were created for evaluating model performance, especially RMSE. Autotuned random forest performed the best (lowest RMSE), so the main focus is on it.
2. Variance Importance plots were created for CART and the 2 Random Forest Models. For CART, only those were kept that had importance value above 0. For the Random forest models, the TOP15 variances are plotted.
3. Partial dependence plots were also created, for our best performing model only: Random Forest (autotune). The TOP 10 variables were calculated, except for `n_days_since` and `n_reviews_per_month`. RStudio kept crashing, after 3 tries and various tuning efforts, it was given up. The calculated mean price is transformed back from log with correction (parameter: `inv.link = exp`). As they also take an eternity to run (especially the continous variables), the results are saved into RData and called directly in the report.