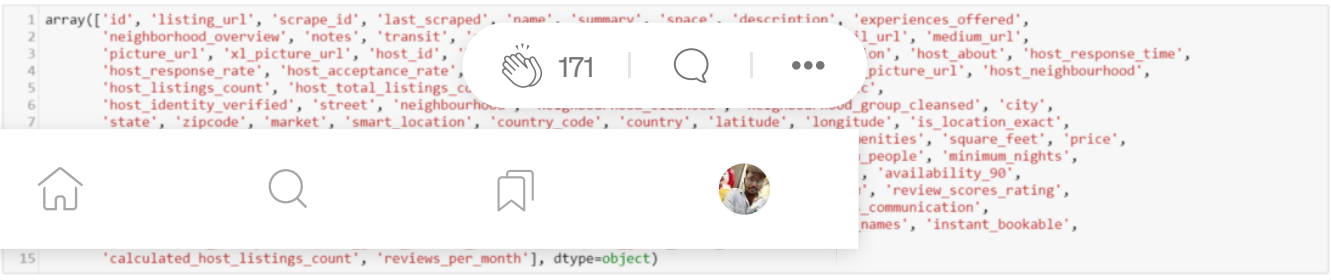
Published in Accenture Interactive Amsterdam

Charles Fried Follow
Nov 30, 2017 · 6 min read · Listen

Airbnb and House Prices in Amsterdam — Part 2

This second and final part looks at applying machine learning techniques to the Inside Airbnb data set. More precisely, we want to predict as accurately as possible the price of a new listing given its attributes.

As with most machine learning projects, the majority of the time is spent understanding and preparing the data-set into a format the machine can work with. In the previous article, we had a very brief look at the data. As you can see below, there's a total of 95 features for 15,181 samples (listings).



All features in 2017 Inside Airbnb Data-set

Given the fact that there are so many features, it's impossible to explore any relationship with price by plotting them on a graph. This is where dimensionality reduction can help us. Concretely, we can use an algorithm called t-Distributed Stochastic Neighbor Embedding (t-SNE) which takes all our numerical features and returns an x and y position based on similarity, which we can use to plot on a graph.

In this case, we've categorised our dataset into four price brackets using a Quantile-based discretisation function, which simply means that we want an equal amount of Airbnb listings in each section. When using this methodology, we don't expect a clear segmentation but instead a gradual one. As you can see below we are presented with some interesting patterns with some colour aggregation, showing that there's clearly some relationship between our features and the price.

Very Cheap: [€18.99, €90.0] Cheap: [€90.0, €118.0] Average:[€118.0, €150.0] Expensive: [€150.0, €3142.0]

This leads us to the following question: **Which features will actually be useful in predicting prices?** Whilst this might seem like an intuitive question, we want to avoid introducing a bias by selecting them ourselves, therefore we'll try to come up with some kind of metric to drive the selection process. We could try to feed everything into a model, however this would most likely

hinder accuracy and most definitely be computationally inefficient.

We'll first take all our numerical features and plot them in a scatter matrix. This allows us to check for any collinearity which could make our model hard to interpret. *Note that the diagonal plots represents the distribution of the feature in question.* There's a slight linear relationship between `beds` and `accommodates,` which is to be expected. The more beds a property has, the higher number of people it can sleep.

Scatter Matrix of Numerical Features

The following visualisation breaks down the data set into individual areas. The radius represents the mean number of ratings which, under the assumption that the rating rate remains equal for every listing, can also be interpreted as the popularity of the area. Then, the mean price of the area is mapped to the width of the slice.

Considering the distance to the centre, its surprising that De Aker ranks first. However, it seems to be a compromise people are willing to make for cheaper accommodation. Following shortly behind in popularity is of course Centrum West, which is also refereed to as de Jordaan, a picturesque area with traditional Architecture and the corresponding price tag.

Polar Axis Pie Chart

Having gained an insight into the data set, we can now crunch some numbers to select some features. There are a number of plain text features such as 'description' which we could use for our model using some Natural Language Processing (NLP), however we'll ignore those for the purpose of this article. Instead we've taken every numerical feature and computed the Pearson's Correlation Coefficients. Bear in mind that this technique only spots linear relationships, so it is not the most accurate choice for feature selection, but certainly better than hand-picking.

Interestingly, the second most valuable feature is `cleaning_fee` which we would mostly likely have overlooked during our potential hand-picking process. *Note that by second, we ignore the first which is price, the features we're trying to predict. The correlation with itself is of course zero. It's worth noting that a negative correlation is just as valuable as a positive one.* `reviews_per_month` *is just as valuable as* `extra_people` *for example. However, in this case neither are particularly interesting.*

Pearson's correlation coefficient

Another common technique to enhance the performance of a learning algorithm is called Feature Engineering. We can combine multiple existing

features to create a brand new one, in this case, we can come up with a `quality` score by multiplying the following features together:

```
quality = number_of_amenities * number_of_reviews *
review_scores_rating
```

The arithmetic here is arbitrary, we could give some weights to each of these or even penalise the value using a negative feature — for example. We just want to make sure neither are equal to zero or it would cancel everything.

Before we can start looking for a model we'll want to convert some categorical features such as area into something called one-hot vectors. This basically turns a word into a number that the algorithm can understand.

Let's start to address the algorithm selection, given we're trying to predict price which lies within a continuous range it is therefore a regression problem (as opposed to classification). This greatly narrows down our choice.

From here we can run some very quick tests on a range of algorithms. We then select the best performing which in our case was also the simplest called Linear Regression.

Our model on average classifies new listings within an accuracy of 23€. Below you can see a residual graph which is simply the error displayed as a scatter plot for each listing in the training and testing set. Despite 71% of all new listings being correctly classified within 40€ there's some large mistakes which we need to take a look at.

Residual Plot on train and test set

In the table below you can see the 5 largest mistakes, they all fall amongst the highest priced listings. Our model is clearly not apt at identifying these. After manually checking the top 20 mistakes we can classify them in three categories:

- The price have been set excessively high by the host, and does not reflect the specification of the apartment (outliers).
- The high number of people the listing accommodates is not fairly reflected by our model.
- The standard of finish is high and cannot be identified using standard features.

Many of the issues listed above can be solved by engineering new features. For example, a luxurious apartment may be be identifiable in its description. Additionally, we could give a higher weight to `accommodates` in our quality score.

I hope you've found this series interesting, if so please give it a few claps to help share it to others. You can also find me on twitter here.

. . .

Github:

Airbnb_Data

Acknowledgement:

Thanks to my colleague Magno Sousa who helped me think this through.

References:

- Inside Airbnb Amsterdam
- Predicting Airbnb Listing Prices with Scikit-Learn and Apache Spark