**Automatic valuation model for Hotel property prices**

**Milestone Report 1**

**Thomas Loeber**

## Summary

The goal of this project is to predict hotel property prices, using a data set of transaction prices merged with a survey containing numerous attributes of the sold properties. Traditionally, properties have predominantly been priced by human experts. However, this is not only expensive, but the result tends to be an extensive focus on the particularities of the property, and an insufficient attention to up-swings and down-swings in the market. While quantitative approaches that model a hotel's sale price as a function of various attributes are also commonly used by practitioners, they commonly use linear regression because of its easy estimation and good interpretability. My goal here is to convince practitioners that machine learning models that are able to model more complex relationships between variables offer a better alternative because of their substantially better accuracy.

## Application

There are two distinct use cases for the resulting models. The most obvious one is investors deciding how much to pay for a specific hotel. The second use is pricing hotel properties for a company's balance sheet. Since the first case requires greater accuracy, we might want to supplement our prediction with the findings from a detailed inspection of the particular property; thus, the second case is the most direct application area of the models developed here.

## Problem statement

Most existing quantitative models of hotel property prices use a simple linear model. There advantage is there easy estimation and good interpretability. While interpretability is good to have, it is attained by sacrificing fit: Most relationships in the world are not linear, and therefore it likely will not be possible in most situations to tell a simple story about what the effect of a particular variable is, because this effect usually depends both on the level of the variable in question, as well as on the value of other variables.

(Note that I do not contend the fact that most phenomena can be well approximated linearly for *nearby* points (first-order Taylor Approximation). An example of a regression

model based on linear *local* approximation is LOESS regression, but ordinary linear regression is based on a linear *global* approximation.)

Of course, linear models are *to some extent* able to model non-linear relationships by including transformations of the original features. Leaving aside the log transformation – which models linear effects on a percentage scale (constant elasticities), this mainly leaves polynomials or dummy variables to measure "real" nonlinearities. However, this strategy has has a number of limitations. Firstly, this does not provide much flexibility of the shape that the non-linear effects may take. For example, the effect of a building's age may not resemble any smooth polynomial; rather haphazardly move up and down depending on how fashionable the building style of the era it was built in is considered now. If, on the other hand, we model this effect by including the dummy variable for, say, each decade, this uses up a lot of degrees of freedom (especially if we do this for all predictors). Furthermore, it forces us to make a lot of decisions about how to divide up a continuous variables into discrete categories, and since this choice in part depends on how much nonlinearity this procedure reveals, it requires fitting many different models. This in turn brings with it the danger of overfitting – unless we reserve a separate test set to evaluate our model, which unfortunately is not standard practice outside of machine learning.
An even more important downside of polynomials – the most common way to model nonlinearities – is that they tend to behave erratically far away from the mean. This is already a problem if we include square terms, but gets even worse if we use higher powers. Note, again, that square terms only allow us to model a relatively simple form of nonlinearity, so including higher order powers is necessary if we wanted to model more complex shapes.
This behavior of polynomials does indeed cause problems for a situation such as ours: While the age of most hotels will fall in a relatively narrow range, there will be a few observations that are much older. Polynomials are not only likely to give us predictions there are far off for these data points, but the fact that ordinary linear regression minimizes a quadratic loss function also means that the fit for the other data points will be compromised in order to avoid creating too extreme outliers for the other points.

While there exist extensions to regression analysis such as LOESS or smoothing splines that are able to model more complex shapes, these also come with a number of assumptions about the distribution of the data. I instead rely on gradient boosting, which is known to provide top performance on anything but huge data sets (where deep learning would be the top choice). The particular variant of gradient boosting I choose is XGBoost, which is particularly accurate and fast.
Of course, we most likely would get even higher accuracy from using an ensemble of different models, but I leave this for future iterations of this project.

The main downside of gradient boosting – as well as any other machine learning algorithm that models complex relationship between the variables – is that it is harder to interpret the results for a human observer. We only get the predictions, plus a measure of which features are most important in generating these predictions. However, by

contrast to OLS, this does not tell us about how much of an impact a one-unit increase of a predictor has, or even whether it has a positive or negative impact (or both, depending on the value of other variables). However, as I argued above, this is not really a fault of the model, but rather is due to the fact that most relationships we observe in the world are simply too complex to be captured by any simple summary.
In order to get at least a rough idea of the effect of specific predictors, I use a resampling technique: I pick the predictor of building age, since this likely has the most complex effect. If we ignore interaction effects, we can compute the variable's average effect by drawing observations and assign a random value for age, and then compute its prediction. If we do this a large number of times, we can then plot the predict it sale price as a function of age.

**Data Sources**

The data used come from two proprietary data sets. One contains records of hotel property sales, but it does not provide very rich information about the attributes of each property. Thus, I supplement it with a second data set consisting of a survey of hotels.
However, the data sets stem from two waves. The older observations already came in merged form (and unfortunately I did not have access to the original data set). For the newer observations, I had to merge the transaction data with the survey of hotel attributes.

**Data Cleaning**

The main challenge was merging the two data sets, because there was no variable (or combination of variables) that could serve as the key. For example, the hotel name could be composed of just the hotel chain's name, or chain name plus city name (concatenated in various ways), or chain plus city name plus other identifier (e.g., landmark or part of the city). Similarly, while the data did contain an address, it was not formatted consistently (e.g., different abbreviations). My solution was to iteratively employ different strategies, first using the original columns (in preprocessed form), and then computing string similarity to match observations.

Another challenge was that some columns contained information about two different variables. (For instance, the fact that a hotel was now closed was appended to the hotel name.) This is particularly problematic if one of these columns is numeric and the other is of type string, because this led to the whole column being imported as a object. In some cases, comments such as whether a hotel is closed or that it has been renovated

were appended to different columns for different observations. I thus used pandas' string methods in order to split up these columns, such that each column contains information about only one variable. I then converted the data to the proper type, if required.

I also had to fix smaller problems such as an inconsistent encoding of missing values. Finding these problems required taking a close look at the data open parenthesis e.g., unique values). One common hint of problems was the inability to convert data to the proper type.
Similarly, I had to concatenate the data from the older and newer wave, which required changing column names so they would match. Since I did not have a data dictionary, this was sometimes challenging, for example because some columns in the older data were transformations of the original data that I had to compute myself before merging the data. In these circumstances, I had to consult with the original authors of the data set to find out what several columns represented with.

**Exploratory Data Analysis**

For the exploratory data analysis, I focus on looking for any evidence that suggests data problems that were overlooked during the data cleaning process. As I explained in more detail in my previous capstone project, I ignore what the plots may suggest about things like which variables are strongly correlated with our variable of interest, or which variables may have a non-linear effect. A primary reason for this decision is that plots usually only are able to the relationships between at most three variables at a time., As a result, the "insights" that we may seem to glean from these plots can be highly misleading, because they be fraught with things such as spurious correlations. A second reason is that my primary machine learning model, XGBoost, does not make many assumptions on the distribution of the data. For example, it is not affected by outliers or non-normal distributions, since the strategy of splitting treats each variable as ordinal. This also allows it to flexibly model non-linear effects.

I start the exploratory data analysis by plotting a heatmap of the correlations between all variables. For the reasons explained above, I do not interpret too much into the correlations. Since I'm using (elastic net) regularization, I don't have to worry about using accurate tool such as bivariate correlation for feature selection. This is because

regularization has a more principled way of shrinking the effect of variables that are unimportant towards zero (or potentially even dropping them, if we include enough of a L1-penalty).

Next, I analyze the minimum and maximum for each numeric variable. Everything looks reasonable, though it might be worth having a domain expert look at the maximum, since it is hard to say for a non-expert how high some of these variables can possibly reach.
I then proceed to plot scatterplots of these bivariate correlations. This can sometimes reveal surprising relationships, such as a variable being truncated, which causes bias for ordinary regression. However, I do not observe anything like that. Again, I don't worry about trying to decipher from these plots whether a variable has a nonlinear effect, because my main model – gradient boosting – is able to automatically deal with this.

For all binary variables, I plot a barchart of their means (in sorted order). This tells us the proportion for which the attribute is present (true). All values fall within the range of zero to one, as they should if everything is correct. I also inspect the actual values to get a feel for the data and to see if they make sense. Again, everything looks reasonable.

Finally, I take a look at categorical variables. I start by inspecting the number of categories for each variable. Three of the six variables have a large number of categories (several hundreds). We need to keep this in mind when performing one-hot encoding, because this will lead to the creation of a lot of additional columns. This is not really a problem for gradient boosting, nor is it for regression as long as we use regularization. However, I will also estimate regression models without regularization for demonstration purposes, as this is commonly done by practitioners. Thus, I will estimate two separate models for each learning algorithm, one containing all categorical variables and one leaving out the three categorical variables that have a lot of categories. (These strategies are often called “unpooled” and “pooled”, respectively.)
For the variables that only contain a small number of categories, I print the name and count for each category. This again helps with getting to know the data set, and reveals the useful information that one of these variables is ordinal. While this is irrelevant for OLS, it can be used to improve the performance of gradient boosting. To do so, we encode this order in terms of ascending integers, so that it can be used by the learning algorithm to make better splits.