Automatic valuation model for Hotel property prices

Final Report

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.

Preprocessing

I ended up dropping all observations with missing values rather than in imputing them. The reason for this is connected to the fact that, as described above, I only had access to the original data for the newer observations. The older observations already came in merged form, and unfortunately all observations with any missing values had already been dropped. Likewise, many variables that were seen as less important had already been dropped.
This is because the data were previously used for regression modeling, so the variables were reduced to a small number deemed most relevant by domain experts. To enable easier modeling, any rows with missing observations were dropped. (This is because the regression models were intended for statistical inference, which requires more complicated imputation than predictive modeling that is typically the goal and machine learning requires. In particular, in order to get correct standard errors, *multiple* imputation needs to be performed.)

I decided to perform the same processing on the newer data. This leaves us with 7,500 rather than close to 11,000 observations, and 32 rather than at least 100 usable features. Note that since gradient boosting does not have any trouble dealing with missing values and is able to use even a high number of features, this decision favors ordinary linear regression. Furthermore, the small size of the data set does not allow gradient boosting to reach anywhere near its full potential. This is intentional, because it creates what is sometimes called a "least likely case": If it can be shown that gradient boosting is preferable to linear regression even in such a case that is stacked against it, it makes a stronger case in favor of it if it prevails even in such a unlikely case. If we preprocessed the data specifically for gradient boosting (kept all possible features) and as we increase the number of observations (don't drop observations with missing values, and in general as the number of collected data increases over time), the superiority of gradient boosting will be more pronounced.

Next, let's talk about the encoding of categorical and ordinal variables. I took different steps here for different models: For gradient boosting, I performed one-hot encoding for real categorical variables, but mapped cortical variables that were in fact ordinal to integers corresponding to the correct order. This works because tree-based models do not distinguish between ordinal and numeric variables. By contrast, for linear regression, I had to perform one-hot encoding for both categorical and ordinal variables.

Leaving ordinal variables as numeric would otherwise amount to the differences between categories being treated as equal, which is usually not the case.

Goal

The main goal of this analysis is to show that models of hotel property prices should use machine learning techniques that are able to model more complex relationships between variables, rather than using the linear regression models that have been traditionally the main workhorse for statistical inference.

A secondary goal of mine is to show that if linear regression models are used – e.g. as a first step or a point of comparison – it still pays to borrow two related techniques from machine learning, regularization and the practice of assessing the performance of the model by measuring predictive accuracy on withheld data.

The predominant way of assessing the trustworthiness of a model in applied statistics is based on verifying theoretical assumptions combined with model checking, e.g. analysis of the residuals. By contrast, it is common practice in machine learning to instead verify the fit of a model through empirical means, namely by withholding part of the data and then assessing the predictive accuracy on the withheld data. This is preferable because it avoids relying on dubious assumptions about the distribution of the data.

Furthermore, in practice model checking is often neglected, or the findings are not presented. Furthermore, the model checking that is done is biased towards not rejecting useless models. Surely this is a major cause of the replica place in crisis in science. For a great discussion of these issues, see [2 cultures in statistical inference].

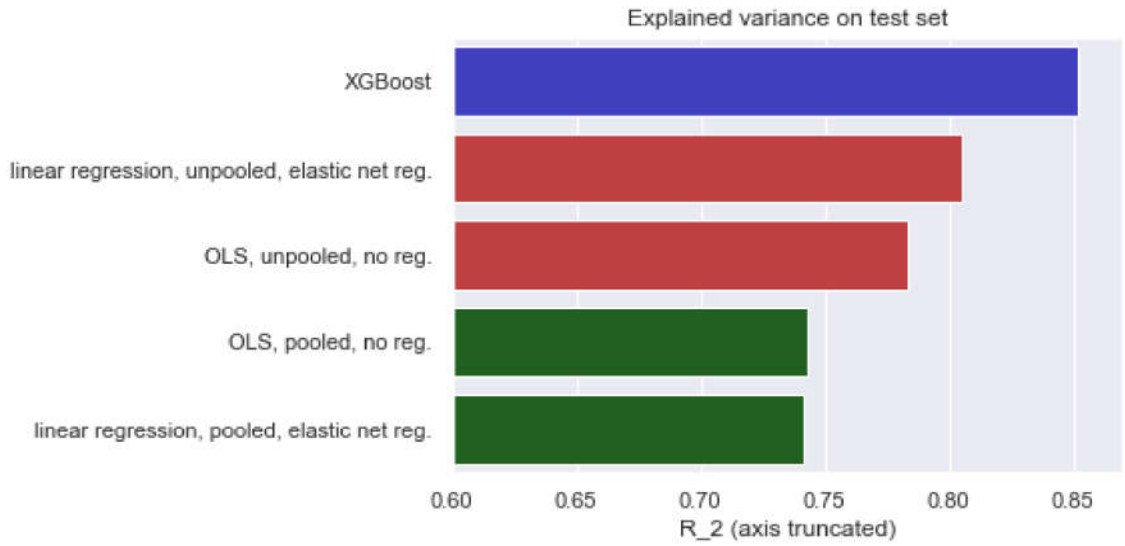
The second technique borrowed from machine learning that should be standard practice in applied statistics is regularization. The purpose is to avoid overfitting our model to the data we train our model on. This problem is usually dealt with an applied statistics by practices such as using the adjusted R_2 for model selection. However, the adjustment done to the R_2 for the last degrees of freedom is not based on any principled way. By contrast, fit statistics based on information theory, such as AIC or BIC, *are* derived from a theoretical framework that tries to adjust for the overfitting to the training set. However, they are not only much less widely used than the adjusted R_2, but are also based on a lot of assumptions about the distribution of our data that are rarely completely satisfied in practice, so it is often hard to say how confident we can be in its prescriptions.

One might think that for our sample size of over 7000, regularly station might not be that important for a simple linear model. However, remember that we had 3 categorical variables with several hundred different categories. If we perform one-hot encoding on these, we thus end up with about 1000 predictors. Since two out of these three variables referred to the location of the property (e.g., the Metropolitan statistical area),

we can expect that it would indeed be y useful if we could include these variables. The only good way to find out if it is worth losing so many degrees of freedom is to estimate the model on part of the data and then seeing which model performs better on the withheld data.

Results

The below table sums up the explained variance on the test set:



Overall, we see that gradient boosting gives us the best performance on new data, with an R_2 of 0.85. By contrast, OLS only achieves 0.78 and 0.74 for the cases unpooled and pooled case, respectively, if we do not use regularization. Adding elastic net regularization improves the performance in the unpooled case from 0.78 to 0.81. By contrast, regularization does not make any noticeable difference for the pooled case. This is not too surprising: In the unpooled case, adding dummies for hundreds of locations and affiliations of the sold hotel led to a proliferation of variables, so regularization is necessary to prevent overfitting. By contrast, in the unpooled case, since we have about 7000 observations and only 37 variables, a linear model seems to have almost reached its capacity: The number of observations is high enough that there is basically no overfitting, and as a result regularization does not yield any noticeable improvement.

Thus, this analysis successfully demonstrates that a state-of-the-art machine learning model is It is preferable to OLS, even in situations that seem to be stacked against it,

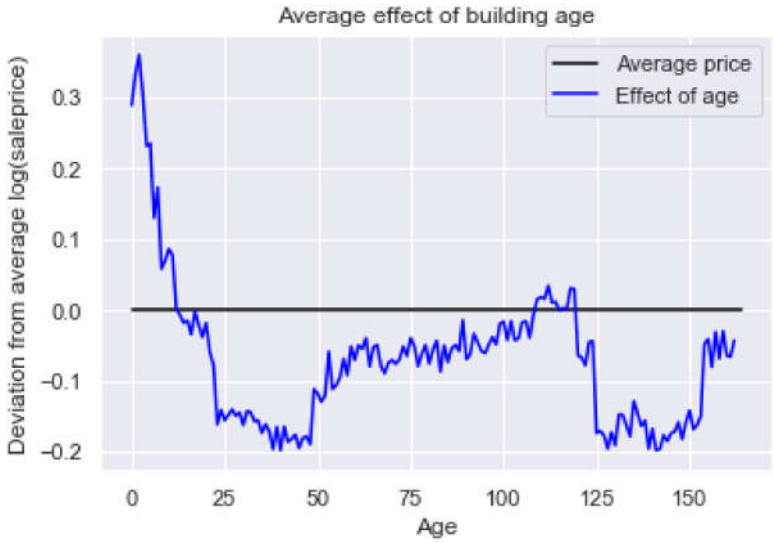
such as a small data set (7000 observations) and not a lot of variables (35). While gradient boosting is more complex to implement, there have been a lot of recent developments that make advanced machine learning techniques more accessible, such as the rise Automated machine learning (AutoML). Furthermore, precisely because machine learning models are more complex, there has been a greater emphasis on developing tools to make model checking easier. In particular, the practice of measuring predictive performance on withheld data offers a simpler and more reliable alternative to applied statistics' heavy use of test statistics that are based on a lot of mathematics as well as often dubious assumptions.

This brings us to the 2nd goal of this analysis, namely to show that, independently of whether more sophisticated models from machine learning such as gradient boosting are adopted, there are also gains to be made from adapting practices from machine learning such as regularization and the use of a separate test set. While using regularization does not always yield noticeable improvements (see the pooled case), it is good practice to always do so, because it is not always easy to say beforehand whether it well. Furthermore, the additional effort involved is small, and it actually saves time in other ways, because we do not have to think as hard about which variables to include and which to exclude. Speaking of which, the reason that regularization did not yield any improvement in the pooled case may also be due to the fact that I'm using a data set in which the variables have already been pre-selected to optimize it for use with OLS. If I had access to the full data set, it is likely that regularization would have an edge, because it is able to intelligently select which variables contain useful information in which mainly contain noise.

Likewise, this analysis demonstrated how important it is to use a second central technique from machine learning, namely the practice of withholding data when training the model and then evaluating the model on these data. Without this, for example, it would have been impossible to say whether the pooled or unpooled model is preferable, or exactly how much better gradient boosting is likely to perform on new data.

Finally, let me say a few words about interpretability. As mentioned above, I argue that it is not the "fault" of more complex machine learning models that they lack easy interpretability; rather, reality is too complex to fit well into simple linear models. Thus, better fitting models will usually be less interpretable for the human mind. While interpretability is good to have, it is not worth sacrificing accuracy and fit.

I illustrate this problem by looking at the effect of age. Gradient boosting models both the nonlinear impact of the feature, as well as the interaction between features. In order to make the effect of age more interpretable, I abstract from the interaction, and simply calculate the effect of age, averaged over all situations. To do so, use a resampling technique: I draw a large number of samples, but randomly assign age values to them. Then, I generate predictions, and plot how the average prediction varies by age.



The resulting graph shows the nonlinear shape that would be impossible to generate by a lower-degree polynomial. This high capacity to adapt to the actual shape of the data is one of the reasons why models such as gradient boosting outperform linear models. As the number of observations grows, this advantage of gradient boosting becomes even more pronounced.