Automatic valuation model for Hotel property prices

Final Report

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.Its advantage is its easy estimation and good interpretability. But while interpretability is good to have, it is attained by sacrificing fit: Most relationships in the world are not linear, and therefore it is not be possible in most situations to tell a simple story that sums up the effect of a particular variable in a single number, 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 still models linear effects on a percentage scale (constant elasticities), this mainly leaves polynomials or dummy variables to model "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, but rather may move up and down haphazardly depending on how fashionable the building style of the era the hotel 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. 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 for a human observer to interpret the results. 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 *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 will use a resampling technique to visualize the effect of a single variable of interest, building age. If we ignore interaction effects, we can compute the variable's average effect by randomly drawing observations, assigning them a random value for age, and computing a prediction for this observation. If we do this a large number of times, we can plot the predicted 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 similar observations.

Another challenge was that some columns contained information about two different variables. (For instance, the fact that a hotel was now closed was simply 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 an object. In some cases, comments such as whether a hotel is closed or that it has been renovated were even appended to *different* columns for different observations. I thus used pandas' string methods in order to split up these columns, such that each column contained information about only one variable. I then converted the data to the proper type, if necessary.

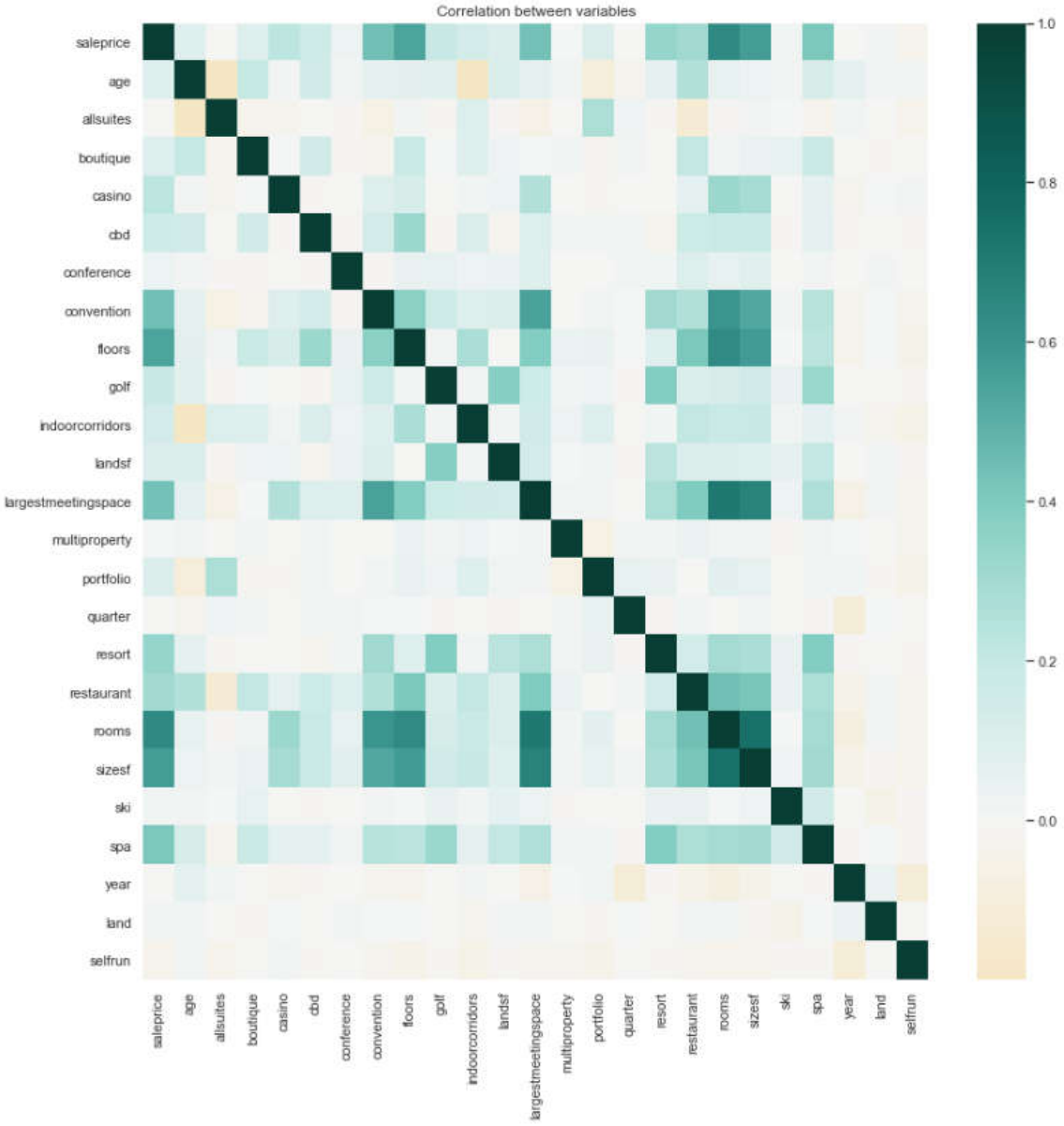
I also had to fix a number of smaller problems such as an inconsistent encoding of missing values. Finding these problems required taking a close look at the data (e.g., unique values). One common hint of problems was the inability to convert data to the proper type.
Similarly, I had to vertically concatenate 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 older data set to find out what several columns represented.

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 target variable (sale price), or which variables seem to have a non-linear effect. One 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 can 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 iterative splits treats each variable as ordinal. This also allows XGBoost 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 unreliable tools such as bivariate correlations for feature selection. This is because regularization has a more principled way of shrinking the effect of unimportant variables towards zero (or potentially even dropping them, if we include enough of a L1-penalty).

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.

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.

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, because if we do not use regularization – as is common practice – it is hard to say beforehand whether the decrease in bias achieved by including the extra variables is offset by the increase in variance brought about by the large increase in the number of predictors. Thus, I will estimate two separate models for each lerning 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

As mentioned in the introduction and problem statement, an important goal of this project is to show that practitioners influenced predominantly by applied statistics should adopt common

practices from machine learning to validate models. (While these approaches may have originated in statistics, the point is that they have only become common practice in machine learning.)

The first such technique is the idea of withholding data (called a test set) when training the model, and then using these withheld data to assess performance. There are two main approaches of how to divide the data into training and test set: We can either split the data randomly, or we can choose a point in time and use all the data originating from an earlier point in time as the training set, and use the later data as the test set. The former approach works best for independent data, and the latter works best for time-series data.
Our data fall somewhere in between: They are not fully independent, because observations that are closer in time are likely to be more similar, even if we take into account all available predictors. (This is often described as having correlated errors.) On the other hand, though, we are not dealing with a true time-series, because we are not following the same units over time. (An example of a pooled time-series would be repeated sales data.) In this true time-series case, we can commonly observe such an extreme serial correlation of errors that this autoregressive process contains more useful information for prediction than the actual features do.
I decided on a compromise: For the *test* set, I use the latest approximately 20% of the data, which cover the last 1.5 years of the sample. This should make sure we are not overly optimistic, as this makes our calculation of the test accuracy more similar to making predictions for use in practice, when we will probably not have a good estimate of the current year's average price yet. (When I calculate the predictions, I simply treat them as if they were from the last year from which we had data, which should give a better estimate than leaving out the information for year completely, because presumably this year's price will be more similar to last year's prize than to the average price of the last 25 years.)
A disadvantage of this strategy is that the newest portion of the data may not be representative of all years, for example because it is drawn only from a particular point in the business cycle. However, this is likely not a severe problem for our case, because the business cycle presumably mainly influences the *level* of the price (captured, in the case of a linear regression model, by the intercept) while the price determinants should stay pretty stable otherwise. To be on the safe side, I do split the data randomly when creating a validation set from the training set when performing cross-validation. The reason that I pick a different strategy here is that we are not interested in the *absolute level* of the validation error here, only the *relative* performance of the different hyperparameters. As a result, it's not a problem that random splitting causes an optimistic bias, because this should affect all hyperparameters values equally.

Let's now talk about missing values. I decided to drop all observations with missing values rather than in imputing them. (This was already done in the earlier Python files that cleaned the data.) The reason for this is connected to the fact that, as described earlier, 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 *columns* that were seen as less important had already been dropped.

This is because the data were previously used for regression modeling. As a result, 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, which is the goal here. In particular, in order to get correct standard errors, *multiple* imputation would need 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 around 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 still prevails. 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 collected more data 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 actual categorical variables, but mapped categorical 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 will perform one-hot encoding for both categorical and ordinal variables. Leaving ordinal variables as numeric would otherwise amount to assuming that the differences between categories are equal, which is usually not the case. Note that I did not perform any standardization, because it is not necessary for the models I use: As already mentioned, gradient boosting does not care about the distribution of the data, because it treats them as ordinal. For linear regression, standardization can be useful to speed up convergence of gradient descent, but our data set is so small that this is not an issue. Thus, I left variables at the original scale, which has the additional benefit of making the coefficients of linear regression more easily interpretable.

Predictive modeling

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

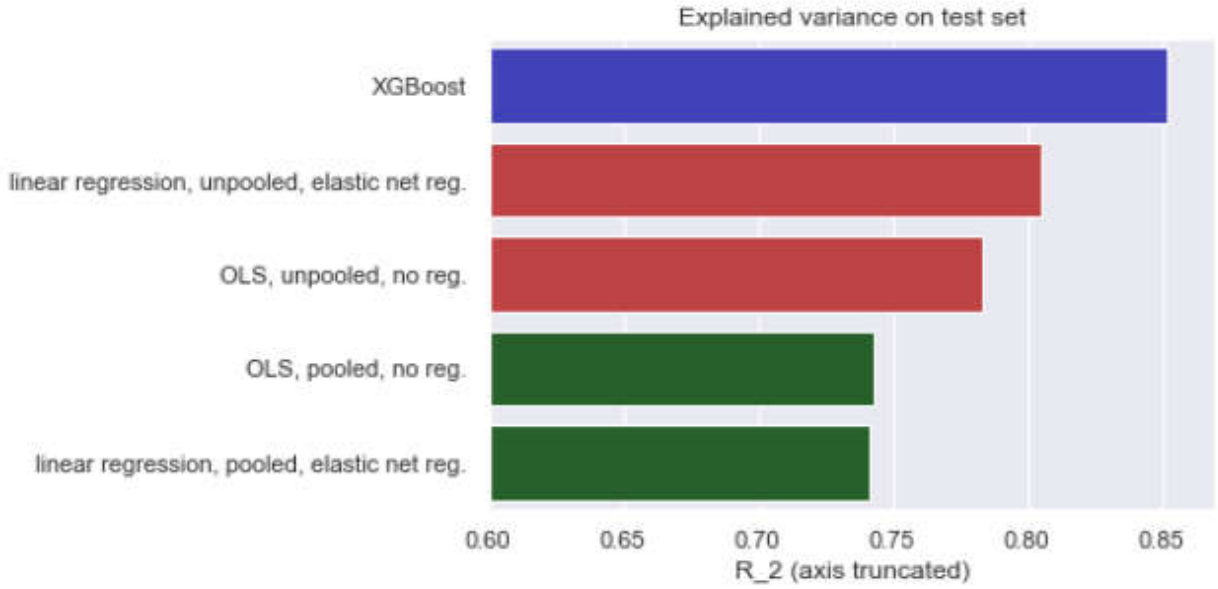
learning, regularization and the practice of assessing the performance of the model by measuring performance on withheld data.

While cross-validation may have originated in statistics, the most common way of assessing the trustworthiness of a model in applied statistics is based on verifying theoretical assumptions combined through 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 more direct means, namely by withholding part of the data and then assessing the model's performance on the withheld data. This is usually 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 with the results. Furthermore, the model checking that is done tends to be biased towards not rejecting wrong models. This is probably a major cause of the replication crisis in many scientific disciplines . For a more in-depth discussion of these issues, see Leo Breiman's famous paper "Statistical modeling: The two cultures," which is available at http://www2.math.uu.se/~thulin/mm/breiman.pdf.

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 in applied statistics by practices such as using the adjusted R_2 for model selection. However, the adjustment of the R_2 for the lost degrees of freedom is not rooted in any principled way. By contrast, fit statistics derived from information theory, such as AIC or BIC, are based on a theoretical framework that tries to adjust for the overfitting to the training set. However, these 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. As a result, 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, regularization might not be that important for a simple linear model. However, remember that we had three categorical variables with several hundred different categories. If we perform one-hot encoding on these, we end up with about 1000 predictors. Since two out of these three categorical variables refer to the location of the property (e.g., the Metropolitan Statistical Area), we can expect that it would indeed be 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.

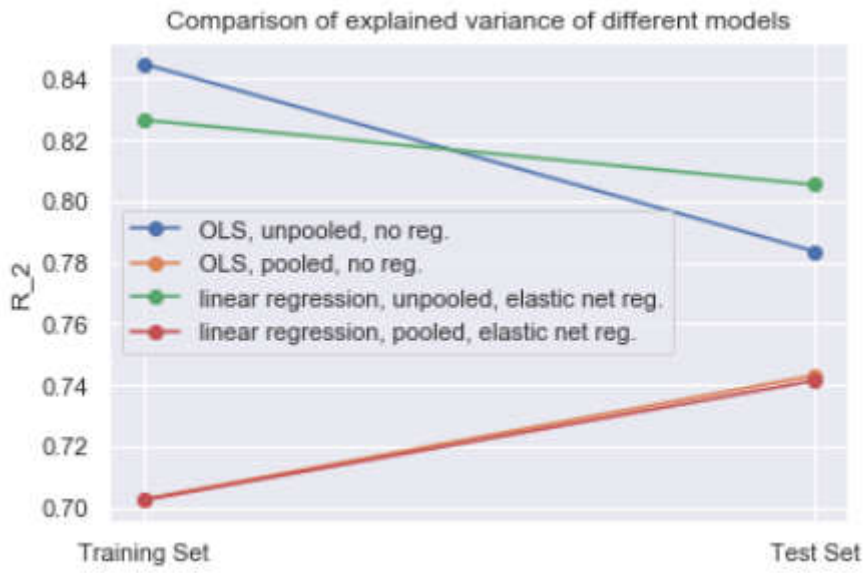
Results

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



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 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 to make it the second best performing model. 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.
Conversely, in the unpooled case, since we have about 7000 observations and only 37 variables, a linear model seems to have virtually 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. A second reason why regularization turned out not to be necessary in this case is 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.

Now let's compare the performance of the linear models on the training and test set to get a better understanding of why the idea of using a separate test set is useful.

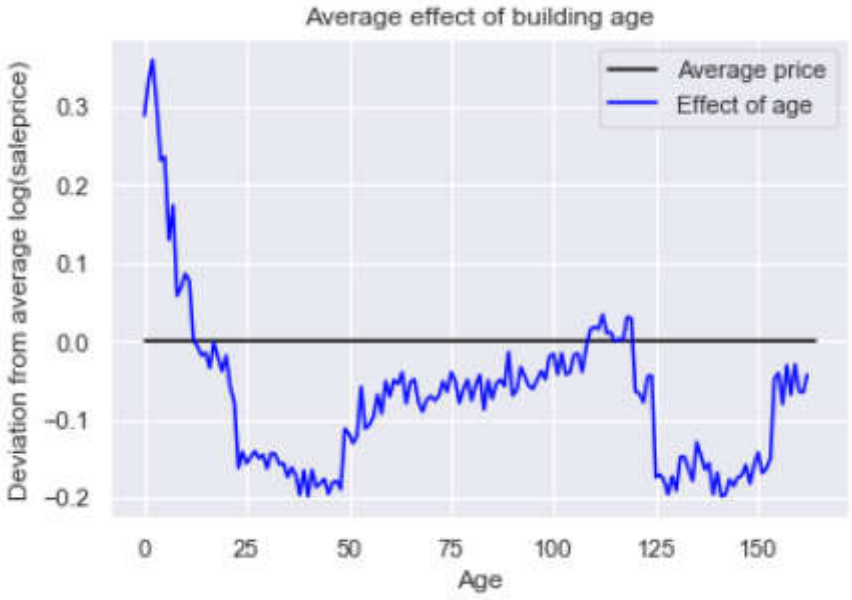


For the pooled model without regularization (blue), we get the typical result: Since it contains a lot of variables (roughly 1000) compared to the number of observations (about 7000), its explained variance is substantially higher on the test set than on the training set. If we add regularization (green line) the performance between training and test set becomes more equalized. (Any remaining difference should be due to randomness.)

If we look at the pooled models (orange and red lines), we see that regularization does not make a noticeable difference. This is due to the fact that these pooled models are characterized by a small number of variables (35) relative to the number of observations (7000). This seems to be more than enough to prevent overfitting. In other words, a linear model has long reached its capacity, and is thus not flexible enough to adapt to the noise rather than the signal in the data.
What is surprising, though, is that these models perform quite a bit better on the test set than on the training set. The difference seems too big to be solely due to randomness (though this may be hard to say intuitively, and we could look at the standard error for the R_2 estimates that cross-validation gave us). Instead, the most likely explanation for how this difference can be so large (in the opposite direction as expected) is that we did not use random splitting of the data, but rather use the later portion of the data as the test set. As a result, the training and test set do not come from the same distribution, and it seems that the test set it is easier to predict. This could be due to the fact that we have much more data for newer years, and so the intercept for the year could be estimated more precisely. Thus, even though we had to treat the new data as if they were from the last year from which we had data, this may still have been more accurate than estimating the effect of year for years for which we only had a few observations.

Finally, let's examine to what extent we can get interpretable results from gradient boosting. 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 inevitably tend to be less interpretable, but this is simply due to the limitations of 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 is able to model both the nonlinear impact of a feature, as well as the interaction between different 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, I 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.

In this way, we could also model feature interactions. For example, we could randomize not only age, but also hotel category. Then we could plot the effect of age for each category. Note, however, that it is only possible to model a small amount of interactions this way, because otherwise the plots become too complicated to generate any insights.

Conclusion

Overall, this analysis successfully demonstrates that a state-of-the-art machine learning model 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 second 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 still good practice to always do so, because it is not always easy to intuitively decide beforehand whether it will. 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.

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 only on these data. Without this it would have been impossible to say, for example, whether the pooled or unpooled model is preferable, or exactly how much better gradient boosting is likely to perform on new data.

Future scope

While we could try different learning algorithms, and combine these into an ensemble to get a presumably small improvement in accuracy, the biggest gains are likely to be had from training the gradient boosting model on better data. As already mentioned, I used data optimized for OLS to show that gradient boosting is preferable even in cases where the odds are stacked against it.

Firstly, it would be very interesting to see how much the edge of gradient boosting increases if we added back the extra features that were discarded. Ideally, we would go back and get the original data for the older wave. However, it might also be sufficient to only use the data we have, and add the extra features from the newer wave while leaving them as missing for the older wave.

Likewise, we should try to increase the number of observations. This could simply be done by not discarding any rows with missing values. (Note that this was done so as to not disadvantage OLS, since there are no good tools available yet that perform a *multivariate* imputation on the training set and then apply the same transformation on the test set. The next version of scikit-learn is scheduled to fix this, though.)

Since location is one of the most important attributes affecting price, we should also focus on engineering better features that measure location at a more fine-ingrained level. For example, since hotel sales are much more sparse than data on residential housing prices, we could use the latter to estimate the desirability of a specific micro-location, and then include this as an additional feature.

Finally, it would be interesting to estimate learning curves, to see if there are any circumstances in which linear regression should still be the preferred model. In this calculation, it might also be interesting to vary not only the number of observations but also the number of features.