House Prices Predictor

Giulio Caputi

Abstract

Here I summed up the work I performed when trying to develop a machine learning model that predicts house prices. The vast majority of time was spent on features engineering (handling missing or wrong data, adding additional variables, and selecting only the relevant features). After having experimented with many different models, my model of choice was a random forest. I have tried both a unique model for the whole test set, and a combination of three different models, each trained on one of the three cities present in the datasets. I eventually opted for the first choice, as it led to significantly-better results (in fact, this way the model was fitted with more training examples, plus the presence of a variable specifically differentiating among the three cities made it easy for the random forest model to divide houses based on their city). I later used this random forest model to make predictions on the provided test set.

Note: This report is mainly a descriptions of the Notebook in `https://github.com/giuliocaputi/HousePricesPredictor`. Please look at the Notebook while reading this document.

Contents

1 Getting the Data Ready to Be Used

The first thing I did, after importing the relevant libraries, loading the dataframe for training, and making sure it did not contain any duplicated value, was to clean it in order to get it ready to be used from training. I started by dropping the outliers (I experimented both with dropping the lowest and highest 5% of prices, and with dropping just the top and bottom 5, and I later opted for this second choice). Next I took care of some problematic values. An example are the *conditions* values, that were originally strings, and that I converted to integers in increasing order of quality. Another example are all the missing values I found, which I generally substituted either with the mean or with the median of the non-null training data. Just dropping them from the dataset was not an option, since they were a lot. I also experimented with building a small linear regression in order to predict some missing values (for example, by regressing $surface$ on *n_bathrooms* and *n_rooms*, one could get, in principle, a good approximation of missing values). The problem was that the dataset did not contain a variable that I could use for this aim, because the None values were abundant for almost all variables. I then turned all the "True" values to a 1, and all the "False" values to a 0 in all the columns that had booleans. For the variables *balcony*, *garden*, and *elevator*, I substituted each null value with a 0 (indicating that the house in question has no balcony, garden, or elevator). The rationale is that, since the data I used were likely obtained through web scraping, if the algorithm did not find information regarding the presence of a balcony, of a garden, or of an elevator in the house data, chances are those things were not present in the house, and so they were not mentioned in the house description. In the data there were houses with a $surface$ value of 0. I decided not to change that, because I noticed that the random forest model (which was my model of choice) was effective in recognizing them as possible errors (in fact my results worsened when I tried to change those values to more-plausible ones). After noticing that in my dataframe there were houses with a construction year greater than 2023, I decided to leave those with a construction year less than or equal to 2025, assuming that those houses have a kind of "projected construction year", and so that information was reliable. For houses that instead had a year value greater than 2025 I assumed there was an error, and so I just set their construction year to 1960 (which is the value I chose also for nulls, since it is the median value, and it is also close the mean, which is 1958). I did something similar with the instances in which $energy\_efficiency$ was greater than 1 million, assuming there must be an error. Also the houses with a very high $floor$ value seemed suspicious, but eventually I ended up only changing the one apparently at floor 56 (located in an area of the Termini neighbourhood of Rome, my hometown, in which houses barely go past 10 floors). Another thing I dealt with were houses with a $floor$ value greater than their corresponding $total\_floors$ value. I set their $total\_floors$ value to the corresponding $floor$ one.

2 Adding Relevant Variables

I added a total of 4 new variables to the dataset.

- *city* and *dist_from_center*

  city is a categorical variable which is set to 1 for houses in Venice, to 2 for houses in Rome, and to 3 for houses in Milan (the order is the same as the order of average prices in the training dataset, meaning that, on average, the houses in Milan present in the training dataset are the most expensive ones, and those in Venice are the cheapest ones). *dist_from_center* is the Euclidean distance between each house and the center of the city it belongs to (St. Marcus Square for Venice, the Colosseum for Rome, and the Dome for Milan). The way I added them was first to compute, for each training datapoint, the Euclidean distance from each of the centers of the three cities, and the lowest of them gave me information about which city that particular house was located in.

- *avg_price_per_location* This feature is the collection of average house prices per square meter in each neighbourhood I identified. In order to create this variable, the first step was to actually divide my datapoints in neighbourhoods. The way I did it was with a k-means clustering of *latitude* and *longitude*. So I started by splitting my training dataset into three portions, one per city, in order to better identify the neighbourhoods. In order to select an optimal value for k (the number of neighbourhoods), I relied on the Elbow method. In other words, I fitted the *latitude* and *longitude* values of each city to many k-means algorithms, each one with a different k, and then I plotted the within-cluster sum of squares as a function of k. I then chose a value for k such that increasing it by one would bring only a negligible reduction in cost. The values I selected were 7 for Venice, 12 for Rome, and 11 for Milan. I later plotted the various clusters to make sure they made sense, geographically speaking. So now it was time to actually divide the houses in neighbourhoods, based on their location (for this I added a variable, *location*, which I later dropped). After having put the data from the three cities back together (and making sure the original order was not altered), I created a dictionary containing the average price per square meter in each one of the 30 neighbourhoods in my training set. I later associated to each house the average price per square meter in its neighbourhood, and I removed the *location* variable, as it was redundant at this point.

- *closeby_poi* This is a variable I computed using the "poi.csv" dataset, containing various information on different points of interest. Since this dataframe had roughly 400 000 observations, I first filtered out all the ones without a *tags.amenity* value (so the ones which did not have any "amenity" label). I later computed, for each of the training observation, the number of points of interest within a radius of 1km from it. I also experimented with computing, for each house, the average distance from the 5 closest points of interest, but this didn't prove to be a particularly relevant variable.

3 Checking for Irrelevant Variables

I later had a look at the correlation matrix of my data, and I noticed that the dependent variable (which is the price of the houses) had a very low correlation (in absolute value) with the variables *balcony*, *conditions*, $energy\_efficiency$, *expenses*, *garden*, *elevator* and $total\_floors$. I didn't drop these variables from my dataset just yet, first of all because correlation is only one of the many ways in which we can assess the relevance of a feature, and secondly because it only measures the linear relationship between any two variables (meaning that if a regressor has a very strong influence on the output, but this influence is not linear, it might still have a relatively low correlation with *price*). A variable that instead I did remove was *proximity_to_center*, as it was likely to capture much of the factors already spotted by $dist\_from\_center$, which is more precise.

4 Building a Model

After all the analyses above, it was finally time to develop an actual model to make predictions. After having experimented with a linear regression, a neural network, and a random forest, I opted for the latter, as it proved to perform significantly better than the other two in this context. The linear regression didn't make particularly precise predictions, probably because the relationships between *price* and the independent variables were not linear, and too complex to be captured by such a simple model. Also neural networks did not prove to be particularly effective, likely because the training data was too little (neural networks generally require more training datapoints than random forests to make precise predictions), or maybe because of great noise in the training dataframe, or maybe because of the way I handled missing values. Instead, the random forest outperformed the other models, and among the possible reasons why this happened I can list its robustness to noisy data, its ability to capture complex relationships, its robustness to outliers,m and its ability to estimate the importance of each feature. Before actually training the random forest I would have later used for predictions, I splitted my train dataframe into a train set (with 80% of the original data) and a test set (with the remaining 20%). I then trained several random forest models on this smaller training set, and I tested each of them on the artificially created test set. This was done in order to select the optimal number of estimators for my random forest, and also to check if the independent variables I had identified previously as the ones with little correlation with the dependent variable were really significant in making predictions or not. Based on several trials, I ended up selecting the value of 300 for the number of estimators of my random forest, and to remove both *expenses* and $energy\_efficiency$ from the training set. So, finally, using the data obtained as explained up to now, I trained a random forest model on the whole training set.

5 Modifying the Test Dataset and Making Predictions

What follows in the code is mainly a repetition of the steps already described above, but this time applied to the test dataset, the one for which I did not have access to the values of *price*, which I had to estimate.