ML Challenge - House Price Prediction

Rocco Totaro - 3159435 - rocco.totaro@studbocconi.it
Kaggle ID: Rocco Totaro

May 9, 2023

Abstract

Following the date exploration and after lightly cleaning the dataset, I did some feature engineering to improve performance relative to the model. I applied features related to the number of bathrooms in relation to the rooms, and features related to location (the distance to the city center and the presence of schools within one kilometer of the dwelling). Following the date exploration and after lightly cleaning the dataset, I did some feature engineering to improve performance relative to the model. I applied features related to the number of bathrooms in relation to the rooms, features related to location (the distance to the city center and the presence of schools within one kilometer of the dwelling); and finally, I inserted the feature of the year of construction. After feature engineering, I ran Random Forest Regression after testing several models. Finally, I continued with Cross Validation, the Principal Component Analysis process and finished with Hyper-Parameters.

Introduction - Data Exploration

I started with a preliminary analysis of the train dataset, going to identify all the various features, datatypes, and non-null counts.

Immediately after performing the preliminary actions, I tried to identify the outliers on the 'price' feature, going on to create an initial idea about the outliers that I deepened later with visualization.

To go about identifying outliers, I made use of two basic visualization tools: the scatterplot for each variable and the heatmap (shown below):

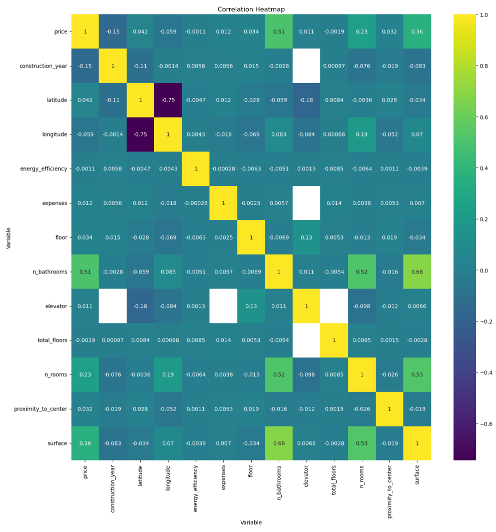

Figure 1: *Heatmap for the identification of outliers*

After that, I found it useful to finish the exploration and visualization phase by clustering these data, initially going to identify the cluster on the train data and then going to do a targeted visualization for individual cities.

After that I went and created a 'missing' dataframe

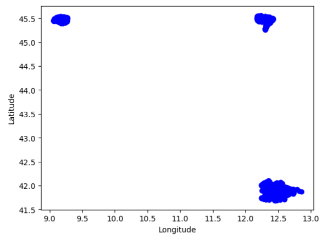

Figure 2: *Scatterplot to identify clusters in the dataframe*

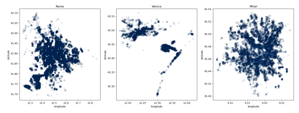

Figure 3: *Scatterplot to identify the distribution in each city*

showing the count and percentage of missing values for each variable in the DataFrame 'train'.

In the first part of the code I calculate the count of missing values for each variable and sort them in ascending order (from lowest to highest). The second part of the code calculates the percentage of missing values for each variable and sorts them in an ascending manner.

After that I went to perform the same operation on the test dataset.

Going to analyze all the missing values in the various features, we can see that the features with the most null data are:

Within the dataframe also, we have data that are found to be not in accordance with reality, specifically, for the feature surface we have some datapoints that are equal to 0.

Or for the feature construction year we have a datapoint that puts it back to +2500; or for the feature floor, we have a building with +31000 floors.

After that I went on to standardize the missing values through the use of fashion and median. I also

Features	Percentage of NaN
price	0.000%
latitude	0.028%
longitude	0.028%
proximity_to_center	0.028%
n_rooms	0.762%
surface	0.957%
conditions	2.654%
floor	4.385%
n_bathrooms	5.217%
construction_year	30.113%
balcony	33.346%
expenses	34.881%
elevator	37.664%
total_floors	39.620%
energy_efficiency	42.976%

Table 1: *Percentage of NaN values for each feature.*

repeated the process for the test dataset.

Having done this, I checked the unique values of the Conditions feature, and after that I went dummies regarding the same feature so as to transform it from a categorical variable into a set of binary variables, in a nutshell I applied the one-hot encoding technique, which is especially useful for making it easier for algorithms to handle such variables.

I applied the same process to the test dataframe and did a check of the missing values; noting that 511 were missing in the feature condition; I then dropped the feature so as to have the cleanest possible dataset.

To finish the cleanup, I deleted all the rows in the train dataframe that had a value equal to 0 as the surface.

In the end, I went from an initial dataset of 46312 datapoints to a final train dataset of 46215.

Feature Engineering

Feature engineering is a critical step in a data science project because it allows the maximum information value to be extracted from the raw data at hand, improving the quality of predictions obtained from machine learning models. In fact, without proper feature preparation, the model may be inefficient and generate unreliable predictions. After testing some new features, these here were the ones that provided the best results in relation also to actual usefulness for the purpose of evaluation by a potential buyer. The features I went on to create were:

Bathroom-to-room ratio

I calculated the ratio of the number of bathrooms to the number of rooms in the house. This function can help us know if there are enough bathrooms compared to the number of rooms, which could be important for potential buyers.

Of course, I also applied that feature to the test data frame.

Location-based features

The idea of this feature was to use the latitude and longitude coordinates to extract location-based features such as distance to popular landmarks, the crime rate in the area, school district, etc. This information can be valuable to potential buyers looking for location-specific features.

To do this, I took latitude and longitude from the center of the cities and initially wrote a function that applies Haversine's formula to go and calculate the distance between two given points and their latitudes/longitudes on a sphere (as indeed the earth is). Haversine's formula is expressed as:

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\Delta\lambda}{2}\right)}\right) \tag{1}$$

where:

- d is the distance between the two points in kilometers
- r is the radius of the Earth (mean radius = 6,371km)
- $\Delta\phi$ represents the difference between the latitudes of the two points in radians
- $\Delta\lambda$ represents the difference between the longitudes of the two points in radians

However, the problem with the Haversine function is that it turns out to be an extremely heavy calculation to compute.

Distance from city center I therefore opted in creating four new features:

- Milan
- Rome
- Venice
- Distance from the center (based on the previous features)

I created a dictionary with the coordinates of the center, filled with the median the latitude and longitude of the missing data. To calculate the distance from the center I went to use this formula:

$$d = \sqrt{(\phi_{house} - \phi_{city})^2 + (\lambda_{house} - \lambda_{city})^2} \tag{2}$$

I then wrote a function that creates three new columns in the dataframe, where the value 1 is assigned if the distance of the house from the center of one of the three cities (Milan, Rome, Venice) is the least among the three, otherwise the value 0 is assigned.

Doing checks on the dataset, I noticed some null values (NaN) which I went to fill in with the classic Pandas methods.

Count of schools within a radius I have defined a function that takes as input the latitude and longitude of a house, created a dataframe 'schools' containing the coordinates of the schools and a radius in kilometers through the use of the dataset 'poi'. The function uses the cdist function of the scipy.spatial.distance module to calculate the distance between the house and all schools in the dataframe.

Next, I select only the schools that are within the specified radius and the number of schools found is returned.

Finally, the function is applied to all rows of the train dataframe using the apply method and a new column is created containing the number of schools found within 1 km of each house.

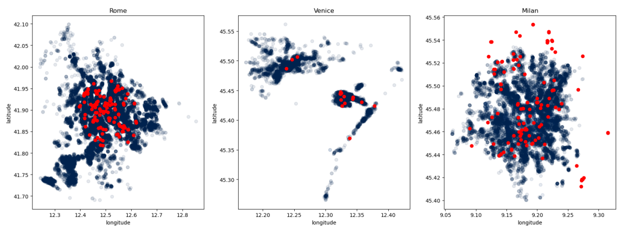

Figure 4: *Scatterplot to visualize the distribution of the schools (red) correlated with the disposition of the houses (blue).*

Age of the house Instead of using the year of construction, I calculated the age of the house by subtracting the year of construction from the current year. This function can be useful because newer houses tend to have a higher price than older ones.

To do this, I set the current year (2023), and went to create a new column that would return the difference between the current year and the construction year.

Model Running

Regarding the workflow for running the model, after testing several models, I found that the best in terms of performance is the Random Forest.

After applying the Random Forest Regressor, I continued with the process of cross-validation and hyperparameter tuning.

Random Forest

Random forest regression is an ensemble machine learning algorithm used for regression tasks. It works by combining multiple decision trees to make predictions, where each tree is trained on randomly selected subsets of the input features and bootstrapped samples of the training data. During prediction, the algorithm averages the predictions of all the individual trees in the forest to obtain the final prediction. The algorithm is robust to overfitting, performs well with high-dimensional and noisy data, and can handle large datasets efficiently.

I created the input data X and the output data y from the train2 DataFrame. The input X contains all the columns except for the price column, while the output y contains only the price column.

Next, I split the data into a training set and a test set.

I split the data so that 33% of the observations are used for testing and the remaining 67% for training. I also set a random seed of 42.

I trained the model on the training set and made predictions of the output values using the test set.

Calculating the MSE between the random forest prediction and $yTest$ yields an MSE of 541255415097.57446.

Cross Validation

I continued by performing cross-validation on a linear regression model to calculate the Mean Squared Error. I set the number of folds to 5, dividing the dataset into five equal parts for this process.

I initialized the linear regression model and created an empty array that will hold the MSE values obtained from each fold.

Next, I introduced the KFold, initialized with the number of folds, set the shuffle parameter to True to shuffle the data before dividing into folds, and set the random seed to 42.

For each fold, I:

- Extracted the training data
- Extracted the test data
- Split the input and output data into train and test sets

In order to train and test the linear regression model.

Then, the model is used to make a prediction on the test data and the MSE is calculated using the MSE function.

The MSE value for the current fold is added to the 'MSE Score' array.

At the end of the loop over all folds, the average MSE value is calculated as the sum of the MSE values divided by the number of folds, and assigned to the variable 'AVG MSE'.

Principal Component Analysis (PCA)

Principal Component Analysis (PCA) aims to reduce the dimensionality of the dataset preserving the most important information. The whole technique is based on finding a new set of orthogonal variables (the principal components) that gets the maximum amount of variance for the dataset.

I have used the PCA class from sklearn.decomposition module, then I calculated the ratios of the variance for each component.

The whole process is based on the Elbow rule, which suggests selecting the number of components at the point where the explained variance begins to level off. This is the point of diminishing returns, where the additional components add little to the total explained variance and may even introduce noise into the model.

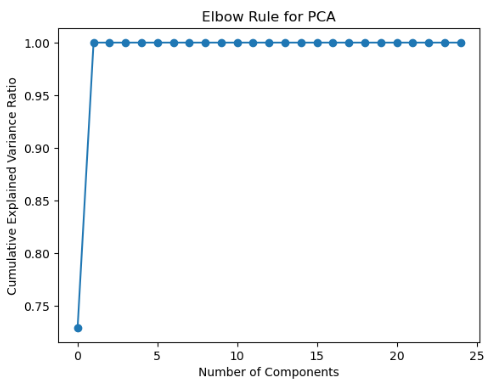

Figure 5: *Visualization of the Principal Component Analysis*

Hyper-Parameters

After this, I applied a Randomized Search Cross-Validation with the goal of selecting the best hyper-parameters to test in the Random Forest Regression.

Next, a Randomized Search Cross-Validation is performed, which combines cross-validation with a random selection of hyperparameters, trying to minimize the error on the training dataset.

Randomized Search is performed over 10 iterations and with a cross-validation of 5 folds. In addition, a random state value is defined to ensure the reproducibility of the results.

At the end of the Randomized Search, the model is trained on the training dataset with the best selected hyperparameters and used to make predictions on the test dataset.

After doing that, I got as best parameters:

- Max Depth: 8
- Max Features: Sqrt
- Min Samples Leaf: 3
- Min Samples Split: 5
- Numbers of Estimators: 64

And finally, I have obtained as final best score of 0.37966934926038276

Final Considerations

While Mean Squared Error (MSE) is a common metric used to evaluate the goodness of fit of a regression model, it may not always be the best metric to use, especially in contexts such as house price estimation. MSE penalizes forecast errors quadratically, which means that larger errors are weighted much more heavily than smaller errors. In the context of house price estimation, this means that forecast errors that lead to oversupply or undersupply can have a very different impact on model accuracy. Additionally, MSE does not take into account the scale of the problem. Therefore, in the context of house price estimation, it is often preferable to use alternative metrics, such as Mean Absolute Error (MAE) or Mean Absolute Percentage Error (MAPE), which give proportional weight to forecast errors and take into account the scale of the problem.

In addition, the MSE does not take into account the scale of the problem. For example, if house prices in the dataset have a wide variation, the MSE may be disproportionately affected by forecast errors in a particularly expensive or economic area.