ML challenge

Sofia Villa 3158972 - Kaggle: Villa Sofia

9th May 2023

Abstract

The challenge has as objective to solve the house price prediction problem. We train a model on certain features of the houses and our goal is to predict the price of the houses of the test dataset as accurate as possible.

Contents

0 Import libraries and load data

As the first thing we install all the required packages (using pip package manager) and libraries we are going to need and we read the "csv" files and store the data in DataFrames.

1 Data Preparing

In this first section we are going to prepare our data, with the goal of obtaining a correct and complete dataset of information that will then be processed and fed to different machine learning models. From now until section 4 we consider only the training dataset, and only in section 5 we will perform all the operations done on the training dataset, even on the test dataset, except for the removal of the outliers of course.

1.1 Utilities Functions

We now define some functions which we are going to use later.

- `NaN_analysis`: with this function we first count the percentage of number of NaNs in each column and also the number of NaNs in each row (the output will be a table with the rows grouped by the number of NaNs). This function is going to be used in the 1.5 Fill NaNs subsection, where we are going to remove or fill this missing or incomplete data points.

- `groupby_count_percentage`: the function tells us the percentage of occurneces for each unique value in the specifies column.

- `hist_subplot`: with this function for each feature we plot both the histogram and the 2D histogram of the train data, we are going to use it later to understand the distributions. It is important to point out that we have to remove the NaNs before, because they are not numerical values and so they can not be plot.

1.2 Point of Interest Analysis

We were given a dataset called point of interest (poi), which could be used as a new feature to help us create a better model. Before we have to understand watch kind of data is, so we perform the NaN analysis on the poi. It results that the majority of the columns of the set are filled with NaNs by more than the 90% . The only 2 columns that can be easily used for the creation of a new feature are "lat" and "lon". We also retain the column "tags.addr:postcode" which we rename "postcode" to attempt to retrieve the postcode of each house in the dataset by studying the postcodes of the points of interest near the house itself.

1.3 Train Dataset Analysis

At this point we perform also the NaN analysis on the dataset (but before we drop the column of the ids, which does not make sense to analyze). From the NaN analysis we can see almost half of the features have more than 30% of NaNs, then in the subsection 1.5 we will deal with this.

1.4 Add Features

We add five new features which can enrich our dataset. This features are created before filling the NaNs, because removing before the NaNs may lead to the loss of important information. This means that later we will have also to fill the NaNs of these new columns.

The first one is the number of point of interest (`n_poi`), which is created considering the latitude and longitude of each house and we get all the poi at a certain radius from the given house.
Two other features are obtained thanks to the estimation of the postcode of each house thanks to the points of interest close to that given house, this time considering a radius ten times larger than that used for the `n_poi` feature. The first of the two features just created by the postcode corresponds to the first three digits of the postcode identifying the province; the second of the two features, on the other hand, corresponds to the last two digits of the postcode identifying the municipality.
The other two features have been created taking in consideration the importance of considering the relationship between the number of bathroom and rooms and also the surface expressed in terms of the extension of a bathroom.

- lambda_ratio: which is the ratio between the number of rooms and bathrooms.
- lambda_m2_per_bathrooms: it calculates the ratio of the surface of an entire house with respect to the number of bathrooms (which can stay in the house), to do so we have made the assumption that the surface of a room is 3 times the surface of a bathroom.

1.5 Fill NaNs

Now we handle the missing values in the dataset, with which the algorithm can not work. What we do with the NaNs is a case by case situation.

1.5.1 Categorical

We first start by working with the categorical values.

- Balcony and Garden: thanks to the previously defined function groupby_count_percentage we can notice that this 2 columns contain only Trues and NaNs, so we assume that the NaNs are False and we then fill the NaN with False.
- Conditions: in this case the NaN represent a small percentage, around 2%, so we decide to fill them with the most frequent value. This choice is based on the fact that given that the other values are not numerical and might not have an accurate average or median value, it makes appropriate to choose the value that occurs most frequently. We are able to maintain the feature distribution by filling NaNs with the most prevalent value.

We then transform the categorical variables in numerical ones, otherwise the algorithm can not work with them. For the boolean variables of garden and balcony is easy, we just transform them in integer numbers (False == 1 and True == 0).
On the other hand for conditions we have to use a one-hot encoding technique, get_dummies which creates numerical variables from categorical ones; where a binary column is made for each category if the categorical feature, and each column is given a 1 or a 0 to show whether or not that category is present for a particular observation.

1.5.2 Numerical

We now deal with the NaNs of the features with numerical values. As a first thing we conduct the NaN analysis (using the function I have described before) and we drop the rows containig more than 10 NaNs, because even if we fill the NaNs values with an algorithm, we will have a biased representation of the data and a too synthetic sample.

For the remaining NaNs we will fill them using the KNNImputer, which is based on the k-nearest neighbors algorithm and it fill each missing value with the average of the 5 closest neighbours. We will do this on all features excluding the price (and id which was drop previously). To do so we divide in x_train (which are the features) and y_train (which is the price) the data set, we perform the KNNimputer only on x _train and then we concatenate together gain x_train and y_train.

2 Data Processing

Our objective in this section is to best process, identify patterns and clean data in order to train the model in the best way possible.

2.1 Data Transformation

We do StandardScaler transformations on the data, but before we plot the histograms and the 2D histograms, which are going to be plotted again after the changes to see the difference in the distribution of the features. Before we separate, as before, the features from the prediction price in respectively x_train and y_train to perform better StandardScaler.
StandardScaler scales and standardizes the data by subtracting the mean and dividing by the standard deviation of each feature. This is necessary because machine learning estimators may behave poorly if the individual features do not roughly resemble typical normally distributed data (for example a Gaussian distribution with a mean of 0 and a variance of 1).

2.2 Drop Outliers

We remove outliers, because their presence may significantly effect the performance of our model by changing the present relevant patterns, in fact by dropping them we improve the quality of the data. The outliers are observation that highly differ from the vast majority of observations in the dataset.
In this case we remove the outliers with the z-score, which let us keep just the observation within 5 standard deviations from the mean, outside 5 standard deviations they are generally considered in fact outliers. We apply the z-score for all the feature in the train set. With this procedure we can see we have remove around 5% of the rows.

2.3 Feature Importance Analysis

We perform the feature importance analysis with the random forest regression model on the train data to know the importance of each feature in predicting the target variable. In case there are features with very low importance we can remove them, but in the case with few features like ours we can also think well not to remove any feature.

3 Training

We loop over a dictionary of regression models and fits each model on the training data. This is useful to then compare the performance of the different regression models and selecting the best one for our task.

4 Evaluation

We create a function to perform an evaluation of our models, the evaluation is done with the train data (used to train the model itself) and not the test data. This choice has been made given the modality of the

challenge. In fact we do not have access to the price of the test data, but we can still see how our model is doing thanks to the 5 daily submissions on Kaggle, but given the limited number of submission with this modality we can still have an approximate idea of how the model is doing. To make the evaluation performed on the same dataset used for training valid, cross evaluation and cross prediction are performed.
For each model we compute the MSE mean and MSE standard deviation and we will submit the model with the lowest MSE, so the model that performs better (in our case the random forest regression model).
We can also visualize the efficiency of the model thanks to the scatter plot, the diagonal line in the plot represents the perfect prediction line. If the points are distributed around the diagonal line, it indicates that the model is performing well.

5 Submission

We preprocess test data with functions that performs a series of steps to add and transform features, fill in missing values, and scale and normalize the data set in preparation for modeling or analysis. We basically apply the transformations done to the train set in the section 1 and 2, except for the removal of outliers, in order transform data to put in input in the model for the predictions. Then we create the csv file we will submit.

6 Tuning

After we have chosen the best model so far, we now want to find the best combination of hyperparameters for the model with tuning.
Hyperparameters are values, chosen prior to training a model, that have an impact on how it behaves and how well it performs. The ability of the model to generalize to new, unexplored data can be dramatically impacted by tuning, making it a vital step in the machine learning pipeline. So we use tuning because a fine-tuned model will probably perform better with unseen data.
In our case we use grid search to tune the hyperparameters for the random forest regression model, which is the model that performs better from the evaluation done in the section 4. We then perform, also on this model, the evaluation test and print its MSE to check if actually there has been an improvement in the performances of the model.

7 Conclusions

The challenge addressed allowed me to develop a model for predicting house prices using regression models. The entire pipeline, typical of machine learning, was performed, commenting on each section in detail. A nan-rich dataset was handled, on which features were added by relying on additional information as well. Finally, it turned out that the random forest model performed best among all the regressors examined.