House Price Prediction Report

UCSD-Extension : MLE Bootcamp

Cathleen Peña

Introduction

In this project, I wanted to create a model that would be able to take in information on houses and predict a house price for the user. This would be useful in cases where certain houses may not have a listed price or it the price may be difficult to find online. With this model, one could enter the information on the features of the house in question and receive an estimate of the house price. For my data collection process, I wanted to find data for house prices in a smaller, more specific region where the value of houses remained similar and didn't vary based on locality, however, this proved difficult as most datasets for cities in America remained very small. Therefore, I thought it best to go with a dataset I found which contained information on houses in Argentina. This dataset had a substantial amount of possible model features and was also within desired size (1000000 rows and 25 columns). Once the necessary data exploration and cleaning was done, the dataset was reduced to about half of it's original size (960,000 rows and 13 columns).

I explored several models including: Random Forest Regressor, XGBoost, Gradient Boosting Regressor, Lasso Regressor and compared them against a dummy mean and median regressor. I evaluated their performance with RMSE, and selected the model that performed best which was Scikit Learn's Random Forest. I then carried out hyperparameter tuning and cross validation with GridSearchCV and saved the best model. I created a file which when ran, will create, train, and save the model. I continued on using Flask Rest API to deploy my model. I made sure to handle any data transformations for categorical variables on the server side so that the user would not need worry about doing all that work. Client side, you simply enter 7 values for the features to send to server side and the server will send back a house price prediction. I've created an html page which can be used to input the values for the house in question if preferred over running the request.py file with the input values.

Data Cleaning & Exploration

For the data cleaning, I handled any type conversions needed, imputed null values, and translated the categorical strings into English. Through exploring the data, I found that the currency of the values in the price column was not consistent as well as

the fact that the prices in the price column were not comparable to each other because some prices reflected the price to be rented for a day, week, month or the price for sale. Therefore, I standardized the prices to all be of the same currency, then standardized the prices so that they would all reflect a monthly price of the house. About half of the features ended up being dropped because majority were null values, or they offered no additional information to feed the model.

Some discrepancies were found in the data such as houses that were outside the target location. The data consisted of 950000 houses in Argentina, 13000 houses in Uruguay, 700 houses in the US, and 90 houses in Brazil. Having houses from many different countries all be in the same dataset, is problematic because the cost of living varies in each country and even city, therefore, not all houses are as easily comparable. After the data has been cleaned and filtered, the last step left was the handle the categorical columns with one-hot encoding.

Modeling

To start off modeling for this project, I explored several models and evaluated their performance. I used RMSE as the primary metric to evaluate the models, as well as MAE and R2 as supporting metrics. The baseline models were Scikit learn's dummy regressor where I used both mean and median regressors. I then trained several models such as random forest regressor, XGBoost regressor, gradient boosting regressor and lasso regressor for which I evaluated. After examining the results, I chose my final model to be the random forest regressor.

	RMSE	MAE	R2
Dummy Mean Regressor	1.860775e+07	112155.846131	-0.000003
Dummy Median Regressor	1.860806e+07	78100.000000	-0.000036
Gradient Boosting Regressor	1.860666e+07	46256.962380	0.000114
XGBoost	1.860635e+07	35083.152344	0.000147
Lasso	1.860710e+07	77189.299512	0.000067
Random Forest	1.860605e+07	5655.306053	0.000180

Next, I carried out hyperparameter tuning with cross validation. This was a long process and took about a day to complete with cv=3. The parameters I chose to tune were: max_depth, min_samples_leaf, and n_estimators. The results of the tuning were surprising in the it showed that the based, untuned model performed better than the tuned one with the base random forest having an rmse of 363768.84 and the tuned one with an rmse of 4871377.79. The best performing model parameters are shown to the right.

```
Out[31]: {'bootstrap': True,
          'ccp_alpha': 0.0,
          'criterion': 'mse',
          'max_depth': None,
          'max_features': 'auto',
          'max_leaf_nodes': None,
          'max_samples': None,
          'min_impurity_decrease': 0.0,
          'min_impurity_split': None,
          'min_samples_leaf': 1,
          'min_samples_split': 2,
          'min_weight_fraction_leaf': 0.0,
          'n_estimators': 100,
          'n_jobs': -1,
          'oob_score': False,
          'random_state': None,
          'verbose': 0,
          'warm_start': False}
```

Deployment

I used Flask's Rest API to deploy my machine learning model. In order to do this, I load the pickled model itself and the pickled model columns. The model columns are necessary for the categorical features that will be input by the user. To build this API, I first create a "Predict" class and more importanty, create the post function that will take in the user's input that they wish to get a price prediction for. The user will not have to worry about categorical values such as the country, province, or property type as the server side will handle all of that. On this server side, it will take in the data for prediction and transform the data into the necessary format for the model to use by one-hot encoding it and then also taking care of any unforeseen feature that might be input. This makes it easier for the user by not making them input an excessive number of features which would be necessary if they had to manually one-hot encode it.Once the data has been transformed properly, it is fed into the model where it will output a house price prediction.

On the client side, the user will simply have to first run the api by running app.py within one terminal, then they can choose to either modify the request.py with their own data and run that file in a separate terminal or should they choose to open up the index.html file, they can also input the values in question there. Both options will post a request to the API and send the data to be input into the model for prediction. The prediction result will show up in the terminal running the api in both cases.

Discussion & Lessons Learned

Throughout this project, there were many lessons learned especially to do with the data. There are several factors of the data used that could have affected the performance of the model. The country Argentina, is too big a region to reasonably predict house prices with this limited amount of data. The locality of some cities might value houses at a higher rate which will mean that hypothetically, the same house with the same exact features could be priced at different prices depending on the city. This could potentially make the model less accurate.

I also found that while exploring the data, although kaggle described the dataset was specifically house prices of houses in Argentina, the data actually contains a column which specifies the country of the house. Once I took a closer look at the values in that column, it revealed that the data contained houses that were in Uruguay, Peru, Brazil, and the United States. This is problematic because the price of housing in different countries can vary drastically based on the cost of living in that country in general but also because of locality value differences— especially between the US and other third world countries.

The final dataset used was not as large as I'd wished it to be. Having more data to train the model on would improve its performance. If the data had been in a more organized format, there wouldn't have been so much empty space in the dataset or extra unnecessary columns. The biggest factors that downsized the data were:

1. Datetime type features could not be used in modeling—3 columns removed
2. Columns l4, l5, and l6 were almost all completely null — 3 columns removed.
3. Approximately 4% of the datapoints were missing the price of the house — 40,000 rows removed.
4. Title and description columns were string columns that did not provide any valuable information that could be use in the ML model — 2 columns removed

The operation_type and price_period feature meanings seemed to coincide a bit and made interpretation of the data, rather confusing. The prices for houses For Sale, are on average, about 200 times more expensive 'monthly' than the 'monthly' average for those houses that are being listed For Rent or For Sublease. This leads me to believe that the price reflected for the houses that are For Sale, is actually just the price of the house and it will be paid little by little on a monthly basis. Essentially, the price_period column seems to not be applicable to houses For Sale. The photo below shows the average price of houses with those certain price_periods and operation_types.

```
In [33]: pd.pivot_table(data=argentina, values='price', index='operation_type', columns='price_period')
```

Out[33]:

price_period	Daily	Monthly	Weekly	other
operation_type				
For Rent	NaN	1285.892401	NaN	1003.004908
For Sale	NaN	241185.580683	NaN	215933.343726
For Sublease	15300.0	3606.735722	10049.655172	2058.518739

Conclusion & Future Work

In the future, I would like to create a script that will scrape the website where the original data had been collected from. Within that script I could create filters to make sure the data collected was only from Argentina and I would also be able to document the features of the data in a more informative manner (as the kaggle description of the features was not easily interpretable). I could then automate this script to pull this data annually, and retrain the machine learning model on the up to date data.

Upon further inspection, after standardizing the price of the house to reflect monthly price for all of the houses by multiplying 'daily' by 30 and 'weekly' by 4, I realized that by multiplying 'weekly' price values by 4 to equate it to monthly price, will

make the average of ‘weekly’ houses be larger than the houses that actually are renting by the month. Consequently, multiplying ‘daily’ by 4 to make it weekly, or 30 to make it monthly, the average of houses selling on a ‘daily’ basis will have a higher average price. This is because I believe people can make a bigger profit if they rent out houses on a nightly basis (they often charge more per night) than if they were to rent it out on a weekly basis. Also, renting houses out on a weekly basis will also still bring in more money per night than if you rent it out on a monthly basis. Therefore, after we standardize the price column, if you look at the average prices of the houses based off the price period, daily > weekly > monthly.

Keeping the price_period columns would actually help the model because although the price period values suggest that these houses would typically only be for rent, most of the houses listed are actually for sale. The average monthly price of the houses that are listed as ‘for sale’ are much higher than those that are being listed as for rent or for sublease. This large difference in house prices makes it difficult for the model to accurately predict the price of the houses that are not For Sale, especially when it comes to listings that are For Rent or For Sublease that are being sold at a monthly basis because the all of the houses that are For Sale are priced monthly which is driving the price of all houses up in the machine learning model since most of the houses in the dataset are For Sale. All of the houses under price periods of either ‘daily’ or ‘weekly’ are for sublease and all houses that are for sale are already under ‘monthly’. So, having the price period column would help to differentiate the prices for those houses that are not For Sale, based on the whether or not they are being rented on a daily, weekly or monthly basis.

Lastly, I would like to make this project easier for anyone to spin up and use. Therefore, the next steps would be to incorporate Docker to package up the environment and make it easier for use on any laptop and secondly, create a website that people could navigate to and input their features to get house price predictions.

In conclusion, through the data collection, data wrangling, modeling and deployment, I was able to carry out this project from start to finish and produce a project where other’s can download and reproduce my model. I was able to learn a lot about how important all the steps in the data and machine learning pipeline are, especially in how to deploy my model with an API.