Nicole Stone

Problem Definition

Finding properties which present a successful margin for a real estate "flip" can be challenging. The best indicator of a profitable real estate investment is a property which can be acquired below 70% of the value at which it could potentially be sold if it were in like-new condition, also referred to as "after repair value" (ARV) [1]. The total cost of the anticipated repairs needs to be accounted for as well, but that requires a person to scope, design, and purchase. That overall renovation cost is then subtracted from the 70% of ARV to provide a maximum purchase price for this investment.

A best-case scenario for a "flip" would be purchasing a like-new home that needs no repairs or updates and can be sold immediately as-is for a profit. This scenario would be a house at 70% or less of the ARV. Therefore, our model should target any homes that are listed at 70% or below the estimated value for that particular property. This problem suggests the use of regression and a possible solution can be using supervised learning to determine an estimate of the resale value of all listed properties. This estimate can then be compared to the actual list price to determine if the property can be acquired for 70% or less of its predicted resale value.

In order to estimate the price at which a home could be sold post-repairs, a machine learning model can be fit on real past sales data for the previous 3 months, as reported by Redfin.com. This estimator will then be used to predict at what price a particular home should be able to sell. Then with new listings, comparisons can be made between the newly listed property's list price and the estimated sale value. Any properties that are listed at or below 70%

of the estimated sale value can then be recommended for further inspection as potential “flip” investments.

Data Analysis

Data Collection

Redfin.com allows for users to download MLS data for up to 350 listings at a time. Bounds can be methodically drawn to include no more than 350 listings in each batch, first for past sales and then separately for current/active listings. For each batch of the selected 350 or less listings, an individual download into a CSV file is conducted. This process is repeated until the entire geographic region of interest is explored and included in a batch of 350 or less listings. In this study, the geographic region is California’s Santa Clarita Valley and surrounding neighborhoods. After collecting all of the CSV formatted files, they are merged using Microsoft Excel such that all past sales are included in one file, ‘past_sales.csv’, and all current listings are included in one file, ‘active_listings.csv’.

Data Cleaning

These files are then loaded into a Panda’s DataFrame object within a Jupyter Notebook environment using Python. The notebook is titled 'Data_Exploration.ipynb' and can be found in the supplementary materials. The initial samples, as downloaded from Redfin.com, include the following features for both current listings and recent sales: SALE TYPE: string, SOLD DATE: string, PROPERTY TYPE: string, ADDRESS: string, CITY: string, STATE OR PROVINCE: string, ZIP OR POST: integer, PRICE: integer, BEDS: integer, BATHS: float, LOCATION: string, SQUARE FEET: integer, LOT SIZE: integer, YEAR BUILT: integer, DAYS ON MARKET: integer, $/SQUARE FEET:

integer, HOA/MONTH: integer, STATUS: string, NEXT OPEN HOUSE START TIME: date, NEXT OPEN HOUSE END TIME: date, URL: string, SOURCE: string, MLS#: string, FAVORITE: char, INTERESTED: char, LATITUDE: float, LONGITUDE: float. Some preprocessing steps will be necessary to create a useful model.

Since some features only contribute irrelevant information for determining a property's value, we will need to remove them. This is to avoid giving the model too many degrees of freedom, as well as to avoid an unexpected pattern to be "detected" by our model based on identifying variables, such as the MLS listing number. The listings will also be checked for any duplicate entries. The features we will want to drop are: ['SOLD DATE', 'ADDRESS', 'STATE OR PROVINCE', 'STATUS', 'NEXT OPEN HOUSE START TIME', 'NEXT OPEN HOUSE END TIME', 'URL (SEE http://www.redfin.com/buy-a-home/comparative-market-analysis FOR INFO ON PRICING)', 'SOURCE', 'MLS#', 'FAVORITE', 'INTERESTED']. None of these features are contributing to the value at which a property is most likely to appraise, but many of them will inject bias into our model because they are intended to help promote the source of these data, or contribute only superfluous information. The 'ADDRESS' feature seems worthwhile to keep at first glance, but 'LATITUDE' and 'LONGITUDE' are both included features in this data set, and 'ADDRESS' is a unique string variable for almost every property.

In addition to removing the features described above, we will want to remove any properties that have a 'SALE TYPE' of 'New Construction Plan' and any properties with a 'PROPERTY TYPE' of either 'Vacant Land' or 'Mobile/Manufactured Home'. These property types are valued in substantially different ways than existing Single Family Residential, Townhouse, and Condo/Co-op properties due to many factors, so we will want to remove them for the

integrity of the model we are constructing. This is because existing Single Family Residential, Townhouse, and Condo/Co-op properties make up a much larger volume of MLS listings, and are also the primary “flip” property types.

Now that we have cleared out any irrelevant or redundant features from our samples, the remaining features are as follows: SALE TYPE: string, PROPERTY TYPE: string, CITY: string, ZIP OR POST: integer, PRICE: integer, BEDS: integer, BATHS: float, LOCATION: string, SQUARE FEET: integer, LOT SIZE: integer, YEAR BUILT: integer, DAYS ON MARKET: integer, $/SQUARE FEET: integer, HOA/MONTH: integer, LATITUDE: float, LONGITUDE: float. This brings us to our complete feature set, however, further preprocessing steps will be necessary to train a regression model.

We will remove any rows which are missing any critical values, such as our target field of 'PRICE'. Then we will zero-fill missing values in our data when it is logically valid to do so, such as zero-filling missing values in the 'HOA/MONTH' since it is reasonable to assume most properties which do not list an HOA payment do so because they do not have one, it would be listed with the tax registrar from past sales if it has been sold previously. So this field should only be null if there is truly no associated HOA for the property or if the property is being sold for the very first time. Now that our data is groomed and void of nulls, we can begin feature engineering.

Solution Implementation

The material covered in this section can be found in the notebooks titled 'Feature_Engineering.ipynb' and 'XGBoost_Model.ipynb' in the supplementary materials.

Feature Engineering

First, we will remove the 'PRICE' feature column to be used as our *y* vector, or target value vector. We can do this at this time since we will no longer be dropping rows from our data sets. We will have two Pandas Series objects, named 'y_active' and 'y_past' which hold all of our target 'PRICE' values.

Since the only MLS data we were able to access contains a mix of categorical and numeric features, we will need to implement a one hot encoding algorithm to the categorical features in our set. We cannot leave these features mixed as categorical and numeric types if we want to use them for regression. Additionally, this algorithm requires that we join the active and past sales data to preserve their identical feature set, then separate them after completing one hot encoding. After completing this step of converting all categorical features to numeric features, we have 106 features in our set.

This leaves us with a very high ratio of features to training samples, so it will be beneficial to conduct dimensionality reduction. After standardizing the data, we will use SageMaker's built-in model for Principal Component Analysis (PCA). Since small percentage differences can mean major financial risk for a region with a median property sale price of $546,300[2], we want to capture as much variance as possible with as few features as possible. 95% of variance was explained for our past sales data with the first 58 principal components, while 99% of variance could be explained with 63 principal components. We can use this new feature set of the top (last) 63 principal components for our price estimator regression model.

Model Selection

Since the data gathered and cleaned for this study make a novel dataset to solve a particular problem of interest, the model selected for training these data is a workhorse algorithm that has garnered much attention and respect in the past five years for its superior regression abilities. There are countless examples online of price estimators being optimized using the XGBoost algorithm, thus it is a prudent choice for trialing this new dataset. This study makes use of Amazon's SageMaker built-in algorithm for XGBoost and the supporting SageMaker resources. The transformed samples will be split randomly into training, validation, and testing subsets.

Since this particular study is not a "big data" project, it will be important to reserve enough samples to adequately test the model, but it will also be necessary to give a considerable sized set of samples for training the model. 20% of the randomized and transformed samples will be reserved for testing, while 80% will be used for training and validating the model. Of the 80% of total samples used for training, 30% of samples will be used as validation during training.

Hyperparameter Tuning

Automated hyperparameter tuning considered the following ranges of each parameter for this study: 'eta': (0.0, 0.5), 'lambda': (0, 1000), 'max_depth': (5, 17), 'num_round': (100, 500), 'min_child_weight': (1, 10), 'rate_drop': (0.2, 0.6).

After hundreds of training jobs, the optimal hyperparameters are as follows for an XGBoost model fit to this particular set of past sales samples: 'rsubsample': 0.8, 'early_stopping_rounds': 10, 'eval_metric': ['mae'], 'eta': 0.4539415243497953, 'gamma': 4.0,

'max_depth': 14, '_tuning_objective_metric': 'validation:mae', 'objective': 'reg:linear', 'min_child_weight': 10.0, 'lambda': 115.80246104977371, 'rate_drop': 0.5, 'num_round': 363.

Results

Benchmark Model

Without the aid of machine learning models, list prices are determined by realtors and then “true” house values are determined by an appraiser using a thorough rubric of property characteristics. Appraiser evaluations cost several hundred dollars for each property; thus, a rough evaluation is completed by the realtor using comparable recent sales or other listings in order to determine a proper list price. The most basic way a realtor determines how much to list a house is to consider current listings and recent sales for the immediate neighborhood of the property they are considering. An average price per square foot of house is determined and then multiplied by the square footage of the property they are listing. This is generally a pretty good starting place, and many realtors list properties solely on this number. A more accurate estimate can be determined by considering lot size, number of bedrooms, and number of bathrooms in comparison to recent sales and then adjusting the list price accordingly. However, for the benchmark in this design, it will suffice to consider price per square foot as an average for the zip code within which the home is located.

Initially, during the proposal design of this project, the metric for a successful machine learning model was intended to be the Mean Absolute Percentage Error (MAPE). However, upon more consideration, the raw value is generally of higher concern for real estate investments than the percentage of an error. Thus, the metric under consideration changed

from this project's proposal, from MAPE to Mean Absolute Error (MAE). The Jupyter Notebook titled 'Benchmark.ipynb' explores the calculated MAE for this dataset, as found by the benchmark method described above, and this notebook can be found in the supplementary materials. As demonstrated in the Benchmark notebook, the MAE for this benchmark method is $83,028.87.

XGBoost Model

The MAE for XGBoost model improves greatly upon the benchmark method, with the MAE for an XGBoost algorithm fit with the parameters above testing in at $32,400.27. In addition to greatly improving the predictive power of what a home's actual value is (when compared to the benchmark method), the XGBoost model was also able to be used to predict "true" values of properties that are currently listed as "for sale" in this market. Using this predicted price, all current listings could be examined against their corresponding prediction to identify any properties that are currently listed as "for sale" that have a list price at or below 70% of the property's estimated value. This allows for clear identification of available properties which fall into the 70% Rule described in the Problem Definition. Figure 1 shows the properties from this dataset which presented the best potential for a "flip" according to these basic rules.

```
flip_listings = pd.DataFrame(full_listings.iloc[flip_potentials][:])
flip_listings.head(6)
```

	SALE TYPE	PROPERTY TYPE	CITY	ZIP OR POSTAL CODE	PRICE	BEDS	BATHS	LOCATION	SQUARE FEET	LOT SIZE	YEAR BUILT	DAYS ON MARKET	$/SQUARE FEET	HOA/MONTH
13	MLS Listing	Single Family Residential	Newhall	91321.0	265000.0	2.0	1.5	NEW4 - Newhall 4	864.0	507360.0	1963.0	3.0	307.0	359.0
47	MLS Listing	Single Family Residential	Canyon Country	91351.0	204999.0	3.0	2.0	CAN1 - Canyon Country 1	1730.0	3500.0	1990.0	12.0	118.0	1257.0
83	MLS Listing	Single Family Residential	Canyon Country	91351.0	164500.0	3.0	2.0	CAN1 - Canyon Country 1	1482.0	3500.0	1998.0	21.0	111.0	0.0
94	MLS Listing	Single Family Residential	Canyon Country	91351.0	227900.0	3.0	2.0	CAN1 - Canyon Country 1	1845.0	3698.0	1990.0	25.0	124.0	0.0
107	MLS Listing	Single Family Residential	Canyon Country	91351.0	172900.0	3.0	2.0	CAN1 - Canyon Country 1	1482.0	3698.0	2000.0	32.0	117.0	0.0
151	MLS Listing	Condo/Co-op	Saugus	91350.0	300000.0	2.0	1.0	PLUM - Plum Canyon	834.0	54817.0	1972.0	73.0	360.0	265.0

We did it! We uncovered six new leads for flipping!

Figure 1. The XGBoost Regression Model Successfully Identifying potential "Flip" Listings.

Conclusions

This project implemented an XGBoost machine learning algorithm to improve upon the inefficiencies of the "cost per square feet for the neighborhood" benchmark method for pricing homes. The inspiration for this project came from the curiosity for a tool that is capable of digesting real estate listings particular to a geographic region of personal interest, generating an estimated value of each property, and then ultimately recommending current real estate listings that may present an opportunity for a "flip" of the property. As demonstrated by the dramatic (256%) decrease in the MAE of the XGBoost model compared to that of the benchmark model, along with the direct "flip" recommendations shown in Figure 1, this project succeeded at solving the problem statement in its entirety.

References

[1] https://www.biggerpockets.com/blog/2014-02-14-70-rule-bible

[2] https://www.zillow.com/santa-clarita-ca/home-values/

[3] https://www.redfin.com/city/17676/CA/Santa-Clarita

[4] Udacity Machine Learning Engineer Nanodegree Course Content