Greater Seattle Area Housing–Sales Price Prediction

Francis Tan

2017-02-27

Abstract

The goal of this project is to predict the sale price of a property by employing various predictive machine learning models in an ensemble given housing data such as the number of bedrooms/bathrooms, square footage, year built as well as other less intuitive variables as provided by the Zillow API.

Data Collection and Preprocessing

Data Collection Process

Although the availability of a public API has made the data collection process simple, there are some limitations that we had to be cognizant of. Our vision was to start with a "seed" property which in turn would collect "comps" or comparables. Comps are simply other properties that have similar features to our seed property. This will provide a buyer an idea of what the value of the property should be.

The first limitation is that the full set of information that we were looking for cannot be extracted from one API endpoint. Zillow does not provide an endpoint which returns property information of comps given a seed property. What it provides instead is one endpoint that returns a list of comp property IDs (Zillow Property ID or ZPID) given a seed property address and a separate endpoint that returns property information given a ZPID. Furthermore, the comp endpoint returns a maximum of 25 comps per seed property. Thus the collection process is divided into three steps:

1. Collect comp IDs given a property address using *GetDeepSearchResults*.
2. Loop through each ZPID, collect 25 more comps for each, and append results to list of the other ZPIDs.
3. Collect property information for each ZPID collected using *GetDeepComps*.

The second limitation is that Zillow has limited the number of calls allowed per day to 1000. This poses a problem if one's intent was to collect a significant amount of data. This limits our collection process further since we had to resort to making two calls. A simple solution was to include a sleep timer of 24 hours when a call encounters a rate limit warning. Although somewhat inconvenient, the solution achieved what we needed to accomplish.

Training Data

The most important element of any data science project is the data itself. This project heavily utilizes data from Zillow, a real estate destination for the internet generation. Fortunately, Zillow provides a public API which provides a convenience to an otherwise tedious task. The table below is just a sample of what the data looks like. We've removed many of the columns to make sure the table fits in the page. This is only to provide an idea of the formatting.

	street	city	state	zip	lastSoldPrice
0	18314 48th Ave W	Lynnwood	WA	98037	315000
1	19011 Grannis Rd	Bothell	WA	98012	353000
2	2625 189th St SE	Bothell	WA	98012	405000
3	719 John Bailey Rd	Bothell	WA	98012	360000
4	5113 212th St SW	Lynnwood	WA	98036	430000

Printing the *shape* attribute shows that we have 2826 observations and 23 columns.

```
>>> training_raw.shape
(2826, 23)
```

Finally, we have the following columns

```
>>> training_raw.dtypes
zpid                   int64
street                object
city                  object
state                 object
zip                   object
FIPScounty            object
useCode               object
taxAssessmentYear     object
taxAssessment         object
yearBuilt             object
lotSizeSqFt           object
finishedSqFt           int64
bathrooms             object
bedrooms              object
```

```
lastSoldDate            object
lastSoldPrice            int64
zestimate               object
zestimateLastUpdated    object
zestimateValueChange    object
zestimateValueLow       object
zestimateValueHigh      object
zestimatePercentile      int64
region                  object
dtype: object
```

Since the goal of this project is to predict the sale price, it is obvious that the *lastSoldPrice* should be the response variable while the other columns can act as feature variables. Of course, some processing such as dummy variable conversion is required before training begins.

Data Processing

The next step is to process or clean the data. First let's take a look at each variable and decide which ones need to be excluded. ZPID and street address logically do not affect sales price and thus can be excluded. Street address may explain some sale price variabilty, however it requires further processing for proper formatting, that is, we must eliminate unit numbers, suffix standardization (Dr. vs Drive), etc. This proves to be a difficult task that is beyond the scope of this project. Further, the effects of this variable is closely related to region. We have chosen to exclude it here but may be worth exploring further in the future. The state variable can also be excluded here as we are keeping the scope of this project to WA only.

```
>>> training = training_raw  # Save original data intact
>>> training.drop(['zpid', 'street', 'state'], axis=1, inplace=True)
>>> training.dtypes
city                    object
zip                     object
FIPScounty              object
useCode                 object
taxAssessmentYear       object
taxAssessment           object
yearBuilt               object
lotSizeSqFt             object
finishedSqFt             int64
bathrooms               object
bedrooms                object
lastSoldDate            object
lastSoldPrice            int64
```

```
zestimate               object
zestimateLastUpdated    object
zestimateValueChange    object
zestimateValueLow       object
zestimateValueHigh      object
zestimatePercentile      int64
region                  object
dtype: object
```

Next we need to see which of these variables need to be converted to factor variables. City, state, zip, FIPScounty, useCode, and region all qualify. One thing to caution when creating dummy variables is the number of unique categories each column has. Large number of categories may be impractical for this project as it requires a significant amount of computing resources.

```
>>> training['city'] = training['city'].astype('category')
>>> training['city'].describe()
count        2826
unique         37
top       Seattle
freq          506
Name: city, dtype: object
>>> training['FIPScounty'] = training['FIPScounty'].astype('category')
>>> training['FIPScounty'].describe()
count      2826
unique        3
top       53033
freq       2127
Name: FIPScounty, dtype: object
>>> training['useCode'] = training['useCode'].astype('category')
>>> training['useCode'].describe()
count             2826
unique               2
top       SingleFamily
freq              2785
Name: useCode, dtype: object
>>> training['region'] = training['region'].astype('category')
>>> training['region'].describe()
count        2826
unique        147
top       Bothell
freq          257
Name: region, dtype: object
```

We can see that none of these variables have an unreasonably high number of unique categories with the exception of region. It contains 147 categories which

may be too high, however, we will assume that our machine can handle this for now.

Finally, let's make the dummy variable conversion. This can easily be achieved using the *get_dummies* function.

```
>>> dummies = pd.get_dummies(training, columns=['city', 'FIPScounty',
'useCode',
...                           'region'], drop_first=True)
... dummies.shape
...
```