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.

Introduction

The Greater Seattle Area housing market has gone through a dramatic price increase in recent years with the median valuation at $609,100, an 11.3% increase from last year and a 4.7% increase forecasted for the next year[1]. Because of the dramatic change that has occurred in a short period of time, it has become increasingly difficult to predict property values. We believe that a machine learning model can predict a property value with reasonable accuracy given the property features and the time aspect (*lastSoldDate*).

Data Collection and Preprocessing

Data Collection Process

The most important element of any data science project is the data itself. This project heavily utilizes data from Zillow, a digital real estate destination for buyers, sellers, and agents. Fortunately, Zillow provides a public API which provides a convenience to an otherwise tedious task. 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

[1]Source: Zillow Data as of February 2017.

call encounters a rate limit warning. Although somewhat inconvenient, the solution achieved what we needed to accomplish.

Training Data

The table below is just a sample of what the training 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.

```
<class 'pandas.io.sql.DatabaseError'>
Execution failed on sql 'SELECT *
       FROM properties
       WHERE lastSoldPrice IS NOT NULL': no such table: properties
```

name 'training_raw' is not defined

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

```
>>> training_raw.shape
Traceback (most recent call last):
  File "< chunk 4 named None >", line 1, in <module>
NameError: name 'training_raw' is not defined
```

Finally, we have the following columns

```
>>> training_raw.dtypes
Traceback (most recent call last):
  File "< chunk 5 named None >", line 1, in <module>
NameError: name 'training_raw' is not defined
```

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 and 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. Finally, 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
Traceback (most recent call last):
  File "< chunk 6 named None >", line 1, in <module>
NameError: name 'training_raw' is not defined
>>> training.drop(['zpid', 'street', 'state'], axis=1, inplace=True)
Traceback (most recent call last):
  File "< chunk 6 named None >", line 1, in <module>
NameError: name 'training' is not defined
>>> training.dtypes
Traceback (most recent call last):
  File "< chunk 6 named None >", line 1, in <module>
NameError: name 'training' is not defined
```

We can see that many of these variables are of type *object*. We'll need to convert these to the appropriate types. Most of these columns, excluding date columns, can be converted to numeric.

```
cols = training.columns[training.columns.isin([
    'taxAssessmentYear',
    'taxAssessment',
    'yearBuilt',
    'lotSizeSqFt',
    'finishedSqFt',
    'bathrooms',
    'bedrooms',
    'lastSoldPrice',
    'zestimate',
    'zestimateValueChange',
    'zestimateValueLow',
    'zestimateValueHigh',
    'zestimatePercentile'
])]
for col in cols:
    training[col] = pd.to_numeric(training[col])
```

```
<type 'exceptions.NameError'>
name 'training' is not defined
```

Now let's convert *lastSoldDate* and *zestimateLastUpdated* to dates.

```
cols = training.columns[training.columns.isin([
    'lastSoldDate',
    'zestimateLastUpdated'
])]
for col in cols:
    training[col] = pd.to_datetime(training[col],
infer_datetime_format=True)
```

```
<type 'exceptions.NameError'>
name 'training' is not defined
```

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')
Traceback (most recent call last):
  File "< chunk 9 named None >", line 1, in <module>
NameError: name 'training' is not defined
>>> training['city'].describe()
Traceback (most recent call last):
  File "< chunk 9 named None >", line 1, in <module>
NameError: name 'training' is not defined
>>> training['zip'] = training['zip'].astype('category')
Traceback (most recent call last):
  File "< chunk 9 named None >", line 1, in <module>
NameError: name 'training' is not defined
>>> training['zip'].describe()
Traceback (most recent call last):
```

```
  File "< chunk 9 named None >", line 1, in <module>
NameError: name 'training' is not defined
>>> training['FIPScounty'] = training['FIPScounty'].astype('category')
Traceback (most recent call last):
  File "< chunk 9 named None >", line 1, in <module>
NameError: name 'training' is not defined
>>> training['FIPScounty'].describe()
Traceback (most recent call last):
  File "< chunk 9 named None >", line 1, in <module>
NameError: name 'training' is not defined
>>> training['useCode'] = training['useCode'].astype('category')
Traceback (most recent call last):
  File "< chunk 9 named None >", line 1, in <module>
NameError: name 'training' is not defined
>>> training['useCode'].describe()
Traceback (most recent call last):
  File "< chunk 9 named None >", line 1, in <module>
NameError: name 'training' is not defined
>>> training['region'] = training['region'].astype('category')
Traceback (most recent call last):
  File "< chunk 9 named None >", line 1, in <module>
NameError: name 'training' is not defined
>>> training['region'].describe()
Traceback (most recent call last):
  File "< chunk 9 named None >", line 1, in <module>
NameError: name 'training' is not defined
```

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.

Let's take a look at our columns now.

```
>>> training.dtypes
Traceback (most recent call last):
  File "< chunk 10 named None >", line 1, in <module>
NameError: name 'training' is not defined
```

Before we convert the categorical columns to dummy variables, let's look at the correlations compared to the sales price.

```
>>> training.corr()['lastSoldPrice'].sort_values(ascending=False,
inplace=False)
Traceback (most recent call last):
  File "< chunk 11 named None >", line 1, in <module>
NameError: name 'training' is not defined
```

As suspected, *zestimate* columns are highly correlated to the sales price. A *zestimate* is essentially Zillow's predicted value. Since we are trying to achieve the same thing in this project, let's not include Zillow's efforts here.

```
training.drop(['zestimate', 'zestimateLastUpdated',
'zestimateValueChange',
               'zestimateValueLow', 'zestimateValueHigh',
               'zestimatePercentile'], axis=1, inplace=True)
```

```
<type 'exceptions.NameError'>
name 'training' is not defined
```

Here are the fininshed columns.

```
>>> training.dtypes
Traceback (most recent call last):
  File "< chunk 13 named None >", line 1, in <module>
NameError: name 'training' is not defined
>>> training.describe()
Traceback (most recent call last):
  File "< chunk 13 named None >", line 1, in <module>
NameError: name 'training' is not defined
```

name 'training' is not defined

name 'training' is not defined

We can see that the median price in our data set is $577,000 which is quite high!

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

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

```
<type 'exceptions.NameError'>
name 'training' is not defined
```

We have 245 columns as shown above, which we can verify by adding the number of unique categories - 1 with the number of non-categorical columns.

Feature Engineering

Although the given set of features are reasonably decent as a starting point, we can do better with feature engineering. One of the most important estimators for a given property's value is its square footage. Of course, comps are never exactly alike so one way to standardize this is to use price per square foot. Let's go ahead and add this new feature to our dataset.

Training

Now that we have prepared the data, we can begin the training process. Since we do not have validation data on hand, we will need to split a portion for out of sample validation. Let's set aside 20% of the data for just that. We achieve this using scikit-learn's *train_test_split* function.