Daren McCulley

CS 591 Project Checkpoint Submission

11/21/2016

**Abstract**

According to a report by the U.S. Census Bureau, US cities pack 62.7% of all Americans into 3.5% of the available American acreage.1 Along with the benefits of population density on this scale come a host of challenges. More established cities, such as New York and London, reached their current size in the middle of the 20th century, but many others, especially in the developing world are still rapidly growing and taking shape.2 Governments and other organizations interested in effectively managing that growth have an increasing supply of data available to draw upon for making informed decisions about how to deal with the challenges they face.

Providing affordable housing options for residents of all income levels is one of those challenges. For some cities, the window on cost-effective and politically-tractable solutions to this problem may have already closed. However, even those cases present an opportunity to autopsy the data. How does a city become one in which a large portion of the labor force responsible for its day to day operation cannot afford to call it home? Through research it might be possible to detect patterns common to metropolitan areas facing a shortage of affordable housing today and offer an analytical tool to people working to prevent a similar fate.

This study considers the Boston housing market between 1985 and 2016; specifically historical assessments, square footage, and geographic locations of over 120,000 residential properties. From the raw data, graphs are constructed for each year to provide a model of the city in order to explore the following theory: by analyzing the properties of such a graph in a given year it is possible to predict with some meaningful accuracy some attributes of this graph for a future year. Ultimately, the hope is that these predicted attributes would be of value to someone combating affordable housing shortages.

**Introduction**

The feature chosen as a target for the machine learning models to attempt to predict is the percentage increase in assessed value, adjusted for inflation, over a five-year period. The motivation for this decision is that property assessments determine property taxes and a high average assessed value in a given cell directly correlates with the income necessary to affordably live there. The housing market has been studied extensively by researchers employing data driven techniques.3, 4, 5 However, these models are complex and rely on data that can be difficult to gather or verify.

By design my approach relies only on data that the city of Boston already produces annually and has a vested interest in doing so accurately. Likewise, the dynamic algorithm for feature generation is simple and easy to understand once the model is built. This study is meant as an exploratory starting point that would be easy to apply to other cities without loss of generality.

**Technical Approach**

*Data Processing*

From the 2016 Boston Property Assessment data6 I obtained the key-attribute “pid” (parcel id). Additionally this dataset contained location data (lat / lng) for roughly 60% of the parcels assessed in 2016. Using “pid” and the BeautifulSoup library for Python, I scraped the city’s online assessment site7 for historical assessment data on each parcel, including property type, address and living area. Google’s Geocoding API along with the address information was used append a latitude and longitude to the other 40% of the dataset. All of this raw data was obtained, stored, and indexed using MongoDB and the PyMongo library for Python.

The raw data was filtered to preserve only condominiums, single-family homes, two-family homes, three-family homes, and four to six-unit apartment buildings. Larger apartment buildings were not considered in this study because a key metric in this model is value per square foot. For a large apartment complex, living area and assessed value do not have the same inherent meaning as they do for the property types list above. Parcels identified as both commercial and residential were not considered for similar reasons. These types of residences should be considered in another study especially focused on the cost of renting rather than the cost of owning.

For the parcels that remain, a value per square foot, adjusted for inflation, was calculated for each year of historical data. Note that this requires a non-trivial assumption. Unlike property type, which is listed for each assessment year, historical living area data was not recorded. If the living area for a given parcel appreciably changed between 1985 and 2016 then the value per square foot is not accurate for some number of those years. By using John Tukey’s8 method to detect outliers, I removed a small percentage of parcels from the dataset based on abnormally high or low values per square foot. However, I saved those parcels in a separate MongoDB collection to later determine the impact this step has on the model and the validity of the outlier label using a less objective approach.

*Modeling*

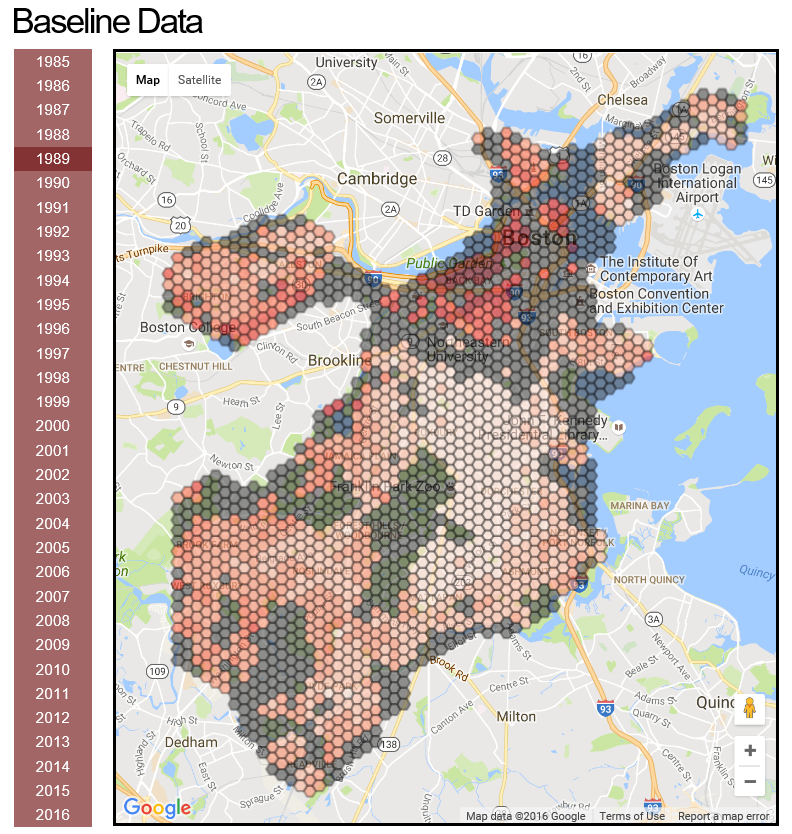
What follows is a discussion of the steps used to construct the graphs modeling the assessed value distribution throughout Boston for each year. A hexagonal tessellation with a side length of one-tenth of a mile, subdivides the map into discrete 0.026 square mile tiles, which I’ll refer to as *cells*. Hexagonal cells have two nice properties: every tile’s center is equidistant and they allow for six possible edges leading to a well-connected graph. I used geospatial indexing and MongoDB’s geoWithin aggregation framework to assign each parcel to its host cell.

A single pass over the parcels in each cell determines the following aggregate cell attributes: total living area, average value per square foot, and number of parcels. Based on the total living area and the number of parcels compared to some threshold for liveness cells are considered *live* or *dead* for a given year. Currently, cells must meet a minimum of 30,000 ft2 and 10 parcels to be considered live. Operating without a threshold allows a small number of a parcels to have an outsized influence on the model.

Cell data is subdivided by year, so that a separate graph can be constructed for each year. There is a node in the graph for every live cell and an edge exists between nodes if the cells they represent are adjacent in the tessellation. Partly because the topology of Boston is so irregular, but also due to the fact that cells are only considered live if they contain enough small residential parcels there is no guarantee this graph will be connected. For example, the city of Brookline and Boston University completely disconnect the live cells in Fenway from the live cells in Allston by as many as seven dead cells. This not a desirable property for the learning algorithm or representative of the underlying concept I’m trying to model.

To address this issue, I introduce a few new cell types beginning with *border* cells. Any dead cell in the complete tessellation that is adjacent to a live cell is labeled a border cell and added to the graph. This effectively expands the original group of live cells by one layer. I repeat this process until the graph is completely connected. In all iterations after the first any dead cell adjacent to a border cell becomes a new border cell. Layers of border cells are added until the graph composed of all live and border cells is completely connected. From the set of border cells, it is desirable to maintain only the cells necessary to connect the graph, which I call *bridge* cells. This is achieved by using the NetworkX library for Python to determine a shortest path between all nearby pairs of live cells and labeling only border cells which lie along one of these shortest paths as bridge cells.

For each bridge cells, I identify all adjacent live cells and label these as *bridge-neighbors*. Again, I use NetworkX to return a shortest path between all closely located bridge-neighbor cell pairs. If the shortest path traverses only bridge cells then a *weak* *edge* is added between that pair. By contrast, a *strong edge* is an edge between two directly adjacent live cells, whereas a *weak edge* is an edge between distant live cells that are separated by a shortest path of only bridge cells. In summation, I artificially grow the graph to completely connect it and then cut away the border and bridge cells to identify where weak edges are necessary. In the end, I am left with a completely connected graph of only live cells featuring strong and weak edges.



A choropleth showing live cells (white to red scale based on value) and bridge cells (grey)

(this is part of a web interface I’m building for the project)

*Algorithm and Prediction*

While the static features used to construct the graph may be significant for their predictive value the benefit of constructing a graph is that one can search for structure and meaningful conclusions by employing dynamic analytical algorithms, e.g. PageRank. The motivation for the algorithm that follows is a belief that when a person is displaced as a result of rising housing prices that person does not go further from their old home than is necessary. Furthermore, like ripples in a pond the displacement of the middle-class by the upper-class may in turn displace the lower-class as the middle-class resettle in their neighborhoods.

In contrast to the considerable time and effort spent obtaining the data and building the model, the algorithm is simple to describe and execute. It is stochastic, with three phases for each epoch. Those phases in order are *flush*, *update flow*, and *outflow*. The amount each cell flushes is a linear function of its total living area. In other words, the amount a cell flushes is a function of its housing capacity. Flow vectors are updated according to the difference in value per square foot, which I’ll simply call value from hereon, across an edge. Finally, all outflows are pushed simultaneously according to the vectors calculated in the update step.

There are three constants, which function as tuning parameters in this algorithm. The first is the flush constant. A high flush constant means that value will rapidly leave the system, whereas a low flush constant allows for value to travel further in the graph before being consumed. The next two are the strong and weak resistances. These values moderate the flow across strong and weak edges respectively. High values result in more of the potential (difference in values) between neighbors turning kinetic and actually leaving a higher value node for a lower value one during each epoch. In short, these three constants determine how quickly the graph approaches an equilibrium state, however they also have a direct bearing on several of the features I collect for machine learning.

**Feature Generation Algorithm**

totalVal = sum(value for each cell)

t = 0 // t: epoch

while totalVal > T: // T: some fixed termination threshold

// flush:

for each cell i:

if remaining value is greater than flush:

i.value = i.value – flush

totalVal = totalVal - flush

else:

i.value = 0

totalVal = totalVal – i.value

// update flow vectors:

for each cell i:

for each strongly adjacent cell j to i:

diff = i.value – j.value

if diff > 0:

i.strong\_flow\_vector.append(diff \* strong\_resistance)

else:

i.strong\_flow\_vector.append(0)

for each cell i:

for each weakly adjacent cell j to i:

diff = i.value – j.value

if diff > 0:

i.weak\_flow\_vector.append(diff \* weak\_resistance)

else:

i.weak\_flow\_vector.append(0)

// outflow

for each cell i:

k = 0 // k: flow\_index

for each strongly adjacent cell j to i:

j.value = j.value + i.strong\_flow\_vector[k]

j.flow\_history[t] = j.flow\_history[t] + i.strong\_flow\_vector[k]

i.value = i.value - i.strong\_flow\_vector[k]

i.flow\_history[t] = i.flow\_history[t] - i.strong\_flow\_vector[k]

k++

for each weakly adjacent cell j to i:

j.value = j.value + i.weak\_flow\_vector[k]

j.flow\_history[t] = j.flow\_history[t] + i.weak\_flow\_vector[k]

i.value = i.value - i.weak\_flow\_vector[k]

i.flow\_history[t] = i.flow\_history[t] - i.weak\_flow\_vector[k]

k++

t++

|  |  |
| --- | --- |
| **Dynamic Features** | |
| Early net flow: | sum of the first 20 entries in flow history |
| Net flow: | sum of the entire flow history |
| Flushed: | total value flushed by this cell |
| Flush ratio: | initial value / total flushed |
| **Static Features** | |
| Initial value | value per square foot before the first epoch |
| Count | number of parcels |
| No. of strong edges | number of directly adjacent cells |
| No. of weak edges | number indirectly adjacent cells |
| Total area | sum of all living areas |
| **Target Feature** | |
| Percentage Increase | (current value – base value) / base value |

Table of the features arising from this data and considered for predicative modeling

**Future Work**

Redefine bridge cells to be the shortest path between bridge neighbors if and only if there is no other path available through live cells between tested pairs. I’m not sure if this will hurt or help the model, but I don’t like that cells on the edge of the graph have a lot of weak edges for no other reason than they are on opposite ends of a concave.

Flip the algorithm to flow first then flush.

Leverage the flow history to do more than just look at early flow or net flow. I will attempt to distinguish between two nodes with a net flow of zero if they got to zero through very different flow histories.

Tune the parameters to produce a more meaningful values for the features.

**Initial Results**

I’m still working on discerning the effects of tweaking my tuning parameters. I also need to come to terms with the effect of the overall trends in the housing market in general during this period. The plots below show cells according to two of the dynamically acquired features resulting from my algorithm (flushed: x-axis and net-flow: y-axis). Each color corresponds to a different percentage increase threshold over the previous five years:

|  |  |
| --- | --- |
| Red: >= 170%  Orange: >= 120%  Yellow: >= 70% | Green: >= 20%  Cyan: >= -30%  Blue: < -30% |

Obviously from 1990 to 1994 the market saw a large downturn.

|  |  |
| --- | --- |
|  |  |
|  |  |

This sort of clustering or striation is common to plots for other years as well. The upper left corner seems to lag the rest of the cells, but is also more extreme. Holding blue longer during downturns and going red and staying hot during upturns. The sparse points in the lower right seem to be bellwethers, i.e. colors move like fronts on a weather map from lower right to upper left. I have a strong suspicion that I should adjust the three tuning parameters in response to the market’s overall trend, but I’m not there yet.

Inline Citations (still require more, and a better tie in)

1. <http://www.census.gov/newsroom/press-releases/2015/cb15-33.html>

2. <http://www.prb.org/Publications/Lesson-Plans/HumanPopulation/Urbanization.aspx>

3. <https://www.geog.uni-heidelberg.de/md/chemgeo/geog/lehrstuehle/gis/helbich_etal_2012.pdf>

4. http://www.governing.com/gov-data/boston-gentrification-maps-demographic-data.html

5. http://link.springer.com/article/10.1007/s101090200086

6. <https://data.cityofboston.gov/>

7. <http://www.cityofboston.gov/assessing/search/>

8. https://en.wikipedia.org/wiki/Outlier#Tukey.27s\_test

Other Related Work (still require tie in to this project)

<http://www.bostonredevelopmentauthority.org/housing>

<http://www.kaynar-rohloff.com/papers/ACSA_Rohloff_Rohloff.pdf>

<http://www.sciencedirect.com/science/article/pii/S0198971506000718>

<http://epn.sagepub.com/content/30/3/523.full.pdf+html>