Report on Capstone Project: Segmentation of houses for sale in Eau Claire, WI

Eric Canton

March 10, 2020

1 Introduction: Business Problem

A young professor couple are moving to Eau Claire, Wisconsin this summer, and want to buy a house. Eau Claire is a small "college city" focused around the University of Wisconsin, Eau Claire campus. There are numerous fairs and festivals in the summer, and avid crosscountry skiing and snowshoeing groups in the winters.

This couple have generous moving funds from their new jobs, so they hire our realtor consulting firm specializing in finding young professionals homes they love in unfamiliar cities. We are the contracted consultants for these clients. Our goal is to provide our clients with an easy-to-use and -understand description of the housing market in Eau Claire. Interactive maps are familiar to most people these days, so we will present our findings to the clients in a Folium map; the locations we find will include links to the property listings, so they can use the websites we scrape to set up viewings of houses and hopefully close on their new home.

1.1 Guiding wishes from the clients

1. The couple hope to have children soon, so definitely want to be near schools.
2. They'd also like for there to be things in walking distance, like parks or downtown areas, but don't want to live downtown.
3. They also want to be aware of local coffee shops where they can go to work remotely.
4. Related to their wish to be able to walk places but not live downtown, we will evaluate the density of venues in the various regions of Eau Claire.

We close this section by noting that while the specific maps produced are mostly of interest to home-buyers looking to move to Eau Claire, WI, the ideas used here can easily be adapted to other locations and segmentation of house listings based on client preferences for venues they want to live near, like "Korean and Italian restaurants" or "botanical gardens".

2 Data Description

After giving an overview of the data we need to perform our segmentation, we will expand some on the steps we use to collect this data. For more details, see the next section, Methodology.

2.1 Data: Overview

The data we need is all related to locations in Eau Claire.

First, we will need to get some listings of houses for sale in the area; to make these most useful, we also want links to the listing, so that our clients can pursue any houses that appeal to them, and we want to provide them with information about the listings like the number of bedrooms, bathrooms, and the price.

Second, we will need information about venues around the houses we find. Since the couple want to have children, we'll look for schools (preschools, elementary, middle, and high schools), parks, and playgrounds. We will look for libraries and bookstores, both for their future kids and because the clients are book lovers themselves. We will also search for coffee shops. Finally, we will estimate the density of the immediate areas around the houses by counting the number of venues within 750 meters.

2.2 Data: Collection

Our general strategy will be to scrape house listing data off of Trulia.com and then make some `search` and `explore` requests to the Foursquare API to get information about venues. More precisely:

1. We will use `selenium.webdriver.Firefox`, running in headless mode, to get the HTML file for Eau Claire house listings on Trulia. These GET requests return 30 listings per page, so we'll get 2 pages.

2. The page source we will pass through the BeautifulSoup `html.parser` to aid our scraping of the contents of that page.

3. Using the `find_all(''a'')` method of our soup object, we can easily find the links within each page we downloaded in the first step. We filter the links by their text: house listing texts all end with "WI" (Wisconsin's state abbreviation), whereas the other links on the page have some other text. The `find_all` method returns a list of these tags and their associated content. Let's look at an example of one of these list elements. There is a lot of HTML describing how to display the little boxes for the houses, some images and alt-text, and so on, so we have pruned the output a bit here to highlight the tags we are most interested in. The original indentation has been preserved here, to give some sense of the nested depth of these tags. (BeautifulSoup makes simple work of extracting these tags!)

```
  <a href="/p/wi/eau-claire/6708-timber-ln-eau-claire-wi-54701--2054962325">
...
<span ... data-testid="property-tag-0">
 <span>
   NEW
 </span>
</span>
  ...
  <div ... data-testid="property-price">
   $379,000
  </div>
    ...
    <div ... data-testid="property-beds">
      4bd
    </div>
   ...
   </div>
    <div ... data-testid="property-baths">
      3ba
    </div>
   ...
    <div ... data-testid="property-floorSpace">
      3,077 sqft
    </div>
    ...
<div ... data-testid="property-street">
  6708 Timber Ln
</div>
<div ... data-testid="property-region">
  Eau Claire, WI
</div>
 ...
  </a>
```

Suppose we have stored this `find_all` list element in the variable `L`. Then `L.text` would return the string:

```
NEW$379,0004bd3ba3,077 sqft6708 Timber LnEau Claire, WI
```

so we can see that the information we're interested in, like price, beds, baths, square footage/size, can all be found in the link text. The address is also in there, but is more difficult to parse because it's not as uniform (as we see from looking at other listings' text). However, it is easy enough to get just these address strings by calling `L.find(''div'', {''data-test-id'' : ''property-street''})`.

4. Once we have the info about each listing loaded into our Listing objects, we'll ask Foursquare for some local venue information.
    - Coffee shops
    - Libraries
    - Book stores
    - Parks
    - Schools
5. Finally, we will pass some "explore" requests to Foursquare, using the response to investigate the areas with dense and sparse venues. These, along with the listings from the previous item, will be used to rate the "venue density".

3 Data: Reporting/Deliverable

The primary means of presenting the gathered data to the client will be through two Folium maps.

1. The first map will have the Listing objects plotted, with HTML tags including the information from Listing member variables. These will be attached to CircleMarkers, colored according to segmentation formed via unsupervised DBSCAN, obtained from Scikit-Learn's OPTICS algorithm. On this map we will also plot the venues of interest mentioned before, color-coded based on their category; for example, the coffee shops will be shown as brown CircleMarkers on this map, and the libraries as teal CircleMarkers.
2. The second map will have the Listings plotted, colored according to the density of venues (of any kind) within 750 meters of the address. The density clustering will be done using K-means, with $K = 3$.

The advantage of presenting the analysis of Eau Claire homes in this way is that the maps are interactive, and are a very familiar format based on pervasive use of websites like Google maps.

4 Methodology

In this section, we expand on the steps we took to create the final product for our clients.

4.1 Importing packages

The importing of packages/libraries we need is accomplished first.

```
import time
import pickle

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib

from sklearn.cluster import OPTICS, cluster_optics_dbscan, KMeans
from sklearn.preprocessing import StandardScaler

# Web stuff
from bs4 import BeautifulSoup
import requests
from selenium.webdriver import Firefox
from selenium.webdriver.firefox.options import Options

# Map stuff
from geopy.geocoders import Nominatim as Nom
import folium
```

The `pickle` package is used only to save the listings and venues we find for re-loading without re-downloading and scraping, which not only takes time due to API requirements (e.g., Nominatim restricts calls to 1 per second, accomplished using `time`) and network availability/speed.

4.2 Python classes for Listing and Venue

We created two Python classes, one called Listing and one called Venue, that store data associated to the houses and venues we find. These are defined as follows.

```
class Listing:
    def __init__(self, addr : str, web : str, ll : tuple, parsed_link_text : tuple):
        self.addr = addr
        self.price, self.beds, self.bath, self.sqft = parsed_link_text
        self.web = web
        self.ll = ll
        self.venues = []

class Venue:
    def __init__(self, name, cat = None, ll = None, web = None):
        self.name = name
        self.cat = cat
        self.ll = ll
        self.web = web

    def get_link(self):
        return "https://foursquare.com/v/" + self.web
```

We populated Python lists of these classes, `Ls` (for Listings) and `key_venues` (for Venues), during our scraping phase (described above in the Data section). These lists were used a number of times throughout our analysis and Folium map creation.

4.3 Machine Learning Techniques

We used two machine learning techniques from Scikit-Learn: K-means clustering and OPTICS.

- We used K-means clustering ($K = 3$) on the densities of nearby venues; the clusters we considered as high, medium, and low densities. We chose K-means clustering here because we had a definite goal of ending with 3 different clusters, and other unsupervised ML strategies like DBSCAN look for natural clusters, with no objective quantity of clusters.

- We used OPTICS for unsupervised segmentation of the Listings we found based on the quantities of Schools, Coffee Shops, Libraries, Bookstores, and Parks/Playgrounds, within a 1.5 mile radius of each house listing. Before running our data through OPTICS, we first used the `sklearn.preprocessing` Standard Normalizer, because some basic EDA revealed that there many parks and playgrounds in Eau Claire, and far fewer of some other kinds of venues, and we didn't want the Parks/Playgrounds category to dominate the segmentation.

  We wanted to use DBSCAN because it allows for discovery of natural, arbitrarily shaped clusters within the data, and we had no preconcieved notions of the quantity of clusters we wanted, so K-means would not be an appropriate choice. However, DBSCAN has two main drawbacks:

  1. the implementation of DBSCAN that Scikit-Learn uses is very expensive, both in terms of processor and RAM consumption.
  2. any DBSCAN implementation necessitates specifying the ε (epsilon) and $MinPts$ parameters to determine the minimum density to be used for clusters.

  While working on another project, we discovered a kind of generalization of DBSCAN called OPTICS. While we still need to specify $MinPts$, no ε parameter is needed and indeed one goal of this algorithm is to allow the user to discover an appropriate ε by ranking the data points based on the ε needed to make a given point a core point, and the relationship/distances to other points. OPTICS also allows one to get around the processor and memory cost of running DBSCAN; while others' research shows OPTICS takes about 1.6 times as long, both implementations in Scikit-Learn currently run in $O(n^2)$. (The other project involved 300,000 data points in three-dimensional space, and took around 30 minutes to complete.) See the next section, Exploratory Data Analysis, for more information on this ranking and the ultimate clustering we found using the "ξ" steepness method.

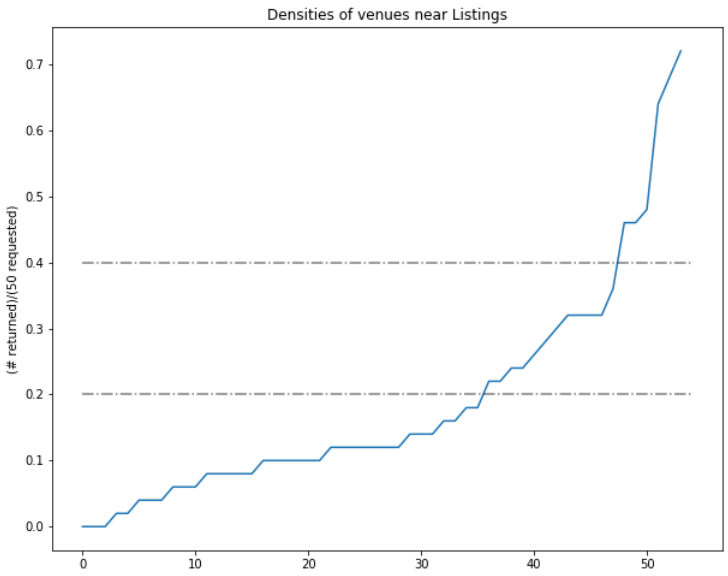
4.4 Exploratory Data Analysis

We used exploratory data analysis in two ways: to analyze the clusters resulting from K-means, and to choose the ε (epsilon) parameter to use in our DBSCAN, based on the "reachability graph" produced from OPTICS.

Analyzing K-means clustering. Before fitting an instance of `sklearn.cluster.KMeans(n_clusters=3)` on our density data, we plotted the densities to estimate the clustering that we hoped to find. The densities were calculated on the number of venues N returned using

```
GET https://api.foursquare.com/v2/venues/explore?...&radius=750&limit=50
```

dividing N by 50. We notice there are two larger jumps, centered around densities 0.2 and 0.4, and indeed these were the break points that K-means ended with, too.



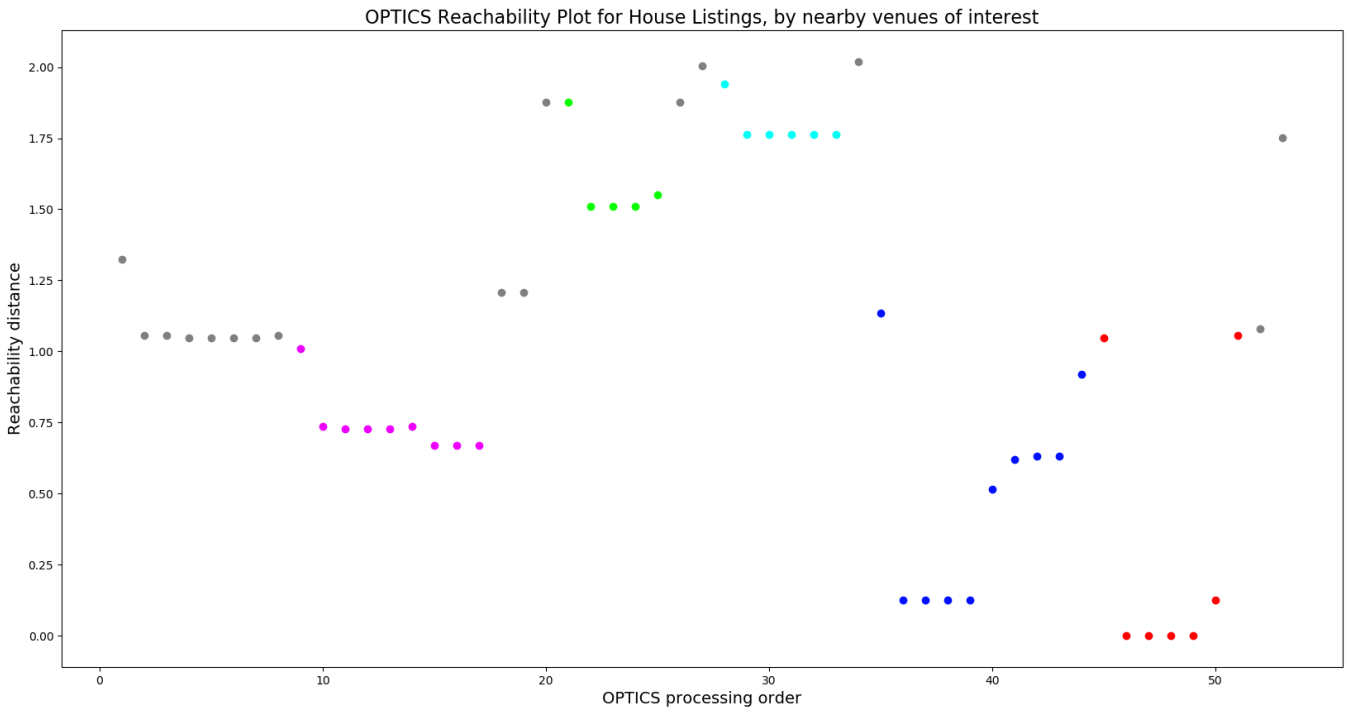
The densities ranged from 0 to 0.72; recall that we computed density out of 50 venues within 750 meters. The dashed lines correspond to break points in the clusters. More precisely, K-means gives:

```
======================
Density class: 0
Min: 0.0
Max: 0.18
======================
Density class: 1
Min: 0.22
Max: 0.36
======================
Density class: 2
Min: 0.46
Max: 0.72
======================
```

Analyzing OPTICS reachability graph. As hinted at in the previous subsection, OPTICS processes the points in a data set in a specific order and measures the so-called *reachability distance* from the current point under consideration to those already processed. This reachability distance essentially captures the radius r one would have to use to cause the current point to be considered in the same cluster as the closest previous point. Thus, if we plot the reachability distances calculated in the order that OPTICS processes the points, then we can visually see the clusters as "valleys" in this reachability plot: remembering that the reachability distance $r(P_j)$ of the j-th considered point P_j primarily involves P_j's distance to P_{j-1}, we see that when the reachability drops it indicates that P_j is much closer to P_{j-1} than P_{j-1} was to P_{j-2}, etc.

The paper [1] originating the OPTICS algorithm proposed the so-called ξ-steepness clustering method, where cluster boundaries are identified on the reachability plots by finding regions where consecutive points are at least ξ% greater or less than the points before. The advantage of this method is that it does not require a fixed threshold be used to decide on what makes a cluster, like one has to do with DBSCAN.

[1] Ankerst, Mihael, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. "OPTICS: ordering points to identify the clustering structure." ACM SIGMOD Record 28, no. 2 (1999): 49-60.


OPTICS Reachability Plot for House Listings, by nearby venues of interest
Reachability distance
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
10
20
30
40
50
OPTICS processing order