Machine Learning Engineer Nanodegree

Capstone Project

Drew Lehe

June 2019

I. Definition

Project Overview

This project uses a dataset of housing sales from Melbourne, Australia, from 2016 through 2017. Melbourne experienced a large property bubble from the late 2000s until 2018, leaving many Australians distraught over rapidly-rising house prices, which then deflated, costing investors billions of dollars. With this project I hope to answer some questions: What creates a high-value neighborhood? Is it close to the city center and densely-populated, or is it remote and sparse? What kind of housing is built there?

By understanding what constitutes a high-value suburb, or a low-value suburb, we can know what housing types to build and create value in every neighborhood. Theoretically, this would ease demand and lower prices in the most expensive neighborhoods, and avert deflation in neighborhoods where price was artificially inflated by a bubble.

I've used machine learning for this problem because I want to build a model that can take in new data and be applied to any city. I don't want to rely on statistics solely for Melbourne, but a program that can be fed any city's data and find groups of neighborhoods.

Problem Statement

Real estate developers and demographers both are interested in tracking neighborhood changes over time, and investigating what caused the change. With clustering, we can group neighborhoods together and find common trends among them. Automating the neighborhood grouping saves researchers time and helps them focus on what is happening in each set of neighborhoods, rather than wondering which neighborhoods are comparable.

In their paper "Cluster Analysis for Neighborhood Change," Reibel and Regelson cluster neighborhoods of Los Angeles to track demographic shifts in neighborhoods. They used KMeans, but I wanted to find a more advanced technique.

I used a Gaussian Mixture Model to cluster, which helped me avoid log-transformations and imputing missing data, and instead let me model the outlying points with their own distribution instead.

Problem 1: Build a clustering tool that groups similar neighborhoods and achieves a high quality for its clusters.

- For this problem I'll use Gaussian Mixture Model clustering to group together similar neighborhoods. A mixture model is good because it allows *multiple* patterns in the data, not just treating all data the same.

For researchers, a common problem is working with sparse data from public websites. If we can predict, with a high degree of accuracy, a feature about housing data, we can help them perform better research in the future. Therefore, I'll also attempt to predict which Council Area a house is in, based on certain features.

Problem 2: Use machine learning to predict which Council Area a house is in.

- For this problem, I'll build a boosted decision tree model (XGBoost) and tune it with cross-validation and grid search.

Metrics

For clustering, I used the silhouette score to examine quality of the clusters. The silhouette score is a measure of how similar the observations in a cluster are. It ranges from -1 (observations are very dissimilar) to 1 (observations are completely alike).

The silhouette value of 1 data point, i:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

Where $a(i)$ is the the average distance between *i* and all other points in the cluster, and $b(i)$ is the *smallest* average distance from *i* to all the points in another cluster (basically, the nearest cluster).

For Council Area prediction, I used F1-score. The F1-score is ideal for a classification model because it weighs the "precision" (number of true positives over the total number of positives) and "recall" (number of true positives over the total number of observations) of your model, which is a smarter metric than simple accuracy. Scikit-learn also has a built-in feature called score_report, which tells me how well I predicted each class (council area).

The F1- score is a weighted measure of your *precision* (p) and your *recall* (r). It is weighted like this to keep it between 0 and 1, like a percentage.

$$F1 = 2 * (\frac{p * r}{p + r})$$

II. Analysis

Data Exploration

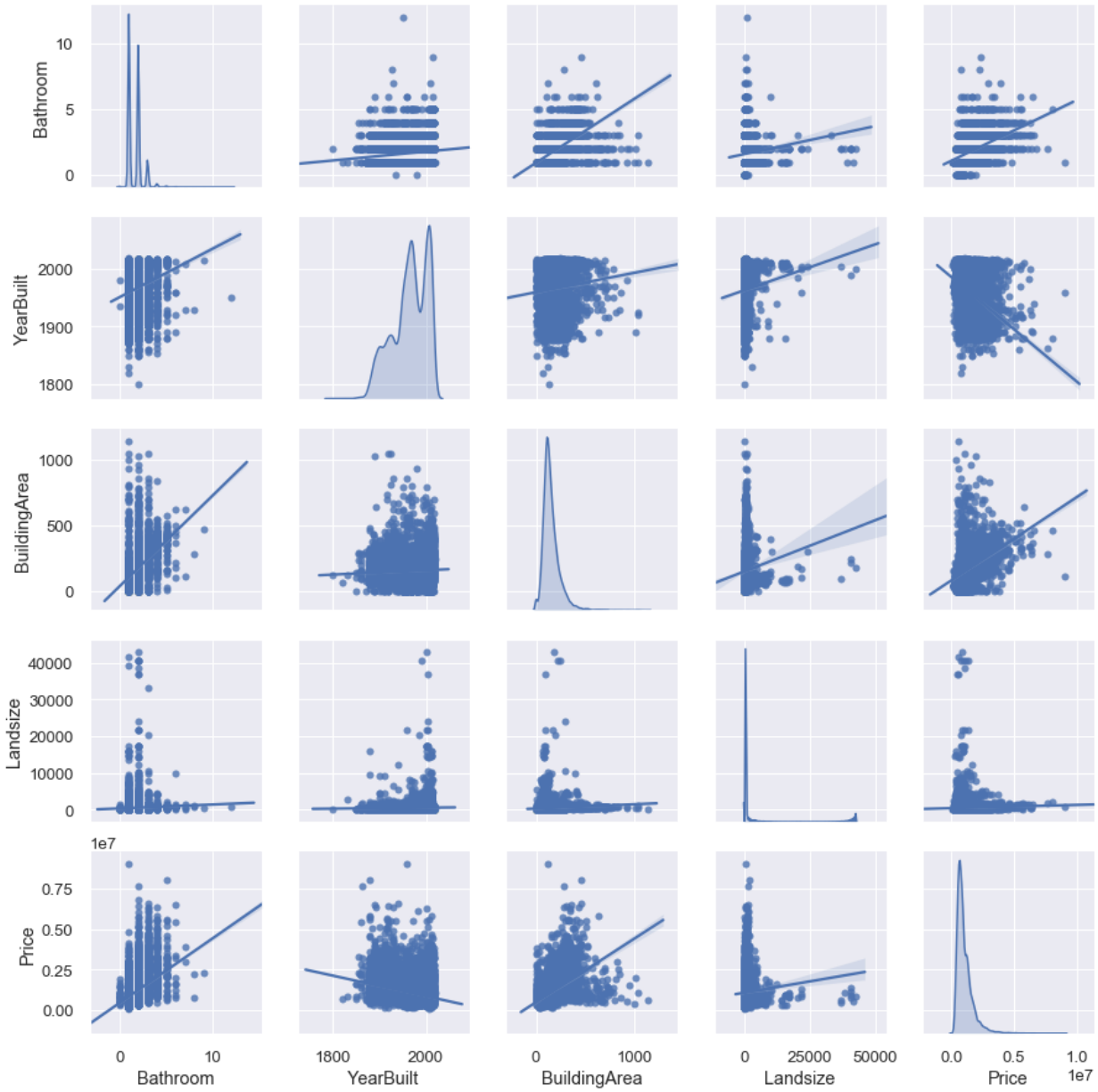
Some observations had erroneously-entered values, which I cleaned out in Excel. Some suburbs only had one observation but I felt that omitting them would make the problem unrealistically easy and the dataset unrealistic. 'Bedroom2' was poorly described and often had a higher value than 'Rooms' so right away I just left it out. It's always bad practice to include a feature in your model you don't understand. Including 'Postcode' sort of defeats the whole point of predicting location or aggregating suburbs, so I omitted that as well.

	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom	Car	Landsize	BuildingArea	YearBuilt
count	34835.000000	2.723100e+04	34835.000000	34835.000000	26619.000000	26610.000000	26109.000000	23028.000000	13730.000000	15545.000000
mean	3.030314	1.049502e+06	11.179354	3115.990613	3.083737	1.624126	1.726531	552.488145	154.796974	1965.328015
std	0.967313	6.380635e+05	6.777694	108.926149	0.977359	0.722772	0.993032	1151.829129	88.104471	36.799981
min	1.000000	8.500000e+04	0.000000	3000.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1800.000000
25%	2.000000	6.350000e+05	6.400000	3051.000000	2.000000	1.000000	1.000000	224.000000	102.000000	1940.000000
50%	3.000000	8.700000e+05	10.300000	3103.000000	3.000000	2.000000	2.000000	520.000000	136.000000	1970.000000
75%	4.000000	1.295000e+06	14.000000	3156.000000	4.000000	2.000000	2.000000	669.250000	187.000000	2000.000000
max	12.000000	9.000000e+06	48.100000	3978.000000	30.000000	12.000000	12.000000	42800.000000	1143.000000	2019.000000

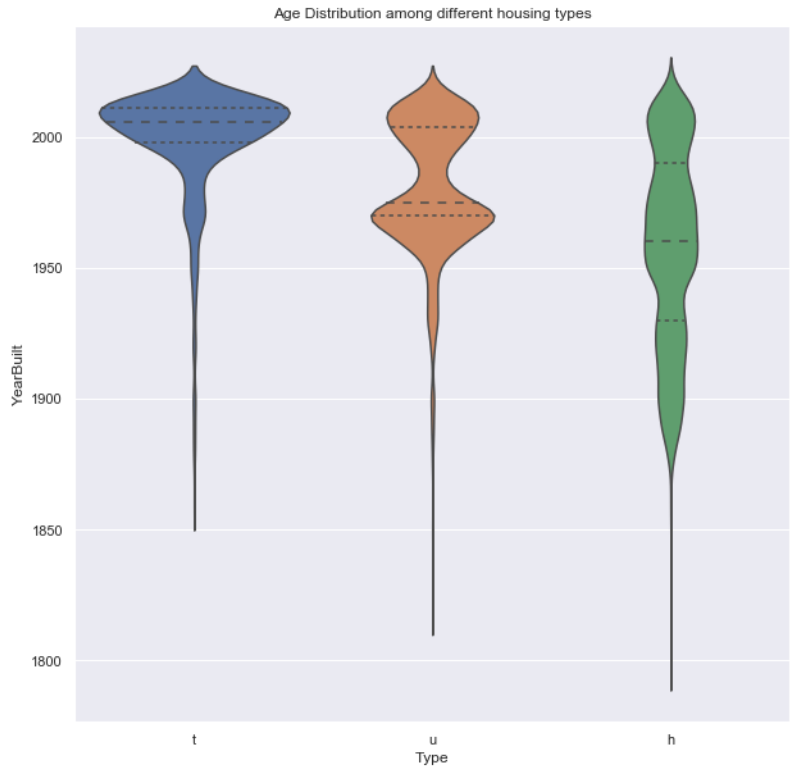
Exploratory Visualization

Right away I created a few correlograms to help me identify outliers and show the relationship between multiple variables. This is an essential part of data cleaning because abnormal points will show up on the graph. Then, as you delete them, new versions of your graph will start to show the "real" pattern you were looking for. It may also tell you that certain features need be log-transformed.

Notice in the plot below, how much of the 'Landsize' value was 0. The challenge of clustering here was working with 2500 '0' values for 'Landsize'. Also notice how 'Price' had a lot of large values. There were so few it was hard to find a pattern, but too many to just delete all of them.



Seaborn's violinplot() function basically combines a boxplot with kernel density estimate, and helps me clearly see the distribution of different classes of the same feature. Incredibly useful for any data scientist doing EDA, it's simply a boxplot with more information. Notice here that townhouse units are overwhelmingly new. So developers have been building a lot of that.



Algorithms and Techniques

Traditionally, a data scientist will use the “elbow method” for determining the appropriate number of clusters in a dataset, looking for a point where the distortion abruptly stops decreasing.

However, if no 'elbow' is present, we must find other methods of choosing a good # of clusters.

For clustering, I used the Gaussian Mixture Model. The Gaussian Mixture Model assume multiple normal distributions are generating the data. It's typically used when your data presents more than one "hump" in kernel density. For this data, I noticed a lot of outliers, but there were too many for me to omit. So I assumed there was a 'double-peaked' distribution for a lot of these features: one distribution for the common observations at the bottom, and another for the rarer observations at the top of the range.

One oft-overlooked advantage of mixture models is that they can, in some cases, help you avoid transforming a feature. I think many beginning data scientists will see a group of outliers and think, "I should omit them," or "I should log-transform this column," but, in reality, they're just part of another distribution that was generating the data.

For city prediction, I used a Boosted Decision Tree classifier (XGBoost) to improve over the benchmark. To improve my prediction quality, I tuned with GridSearchCV and tested a combination of different parameters (max_depth, learning_rate, gamma).

Decision trees on their own generally perform worse than linear regressors, but Boosted Trees are a famously powerful predictor. The basic idea is that it combines multiple "weak learners" to create one "strong learner": Boosting is when the model makes multiple decision trees with part of the available features, and "updates" its guess based on the residuals. At every "boosting round," it tries to minimize the residuals. XGBoost has a "learning rate" parameter that I think of as a "pinch of salt" parameter: XGBoost will update its guess based on what it learned from a boosting round, but the learning rate tells it to only learn so much. Adjusting too much based on what it learned from one round may cause it to overcorrect.

Benchmark

I benchmarked my clusters with Hierarchical Clustering. Hierarchical Clustering is better for datasets with a small number of observations (I had a little over 340 suburbs listed). I tried different cluster numbers to see which gave the best silhouette score, and used that as my benchmark (34.5%):

```
For n_clusters = 2 The average silhouette_score is : 0.38817939364000056

For n_clusters = 3 The average silhouette_score is : 0.3451268065823988

For n_clusters = 4 The average silhouette_score is : 0.30471399116256653
```

```
For n_clusters = 5 The average silhouette_score is : 0.28532489721201054
```

I benchmarked the city prediction with a K-Nearest Neighbors model. I'm particularly interested in KNN because I think of it as an intermediary step between clustering and predictive machine learning. Right away it tells us, "How good of a prediction can we make by just associating an observation with its closest neighbors in p-dimensional space?"

```
Accuracy (F1) score: 0.20854922279792745
```

Additionally, I used a Boosted Decision Tree classifier to improve over the naked decision tree function in scikit-learn. Then, to improve my prediction quality, I tuned with GridSearchCV and tested a combination of different parameters (max_depth, learning_rate, gamma).

III. Methodology

Data Preprocessing

The dataset from Kaggle was fairly clean. However, some values were incorrectly entered. To find abnormal values, first I created a correlogram with seaborn's pairplot() command. Right away this showed bad distributions or outlying points. Often I could use df['feature'].max() or .min() to find a bad value.

Notably, since this data is publicly available from real estate sites, I could often find a house by entering its address or square footage in Google. Pulling up a webpage with photographs and details of the house was useful in deciding if the observation was relevant to analysis. If a house is listed with a 36-car garage, I can google it and find out it's not a house and should not be in the dataset. Some houses had lot sizes abnormally large for a residential neighborhood, so I could look them up and find out they were actually rural areas, too far from town to be considered a Melbourne suburb.

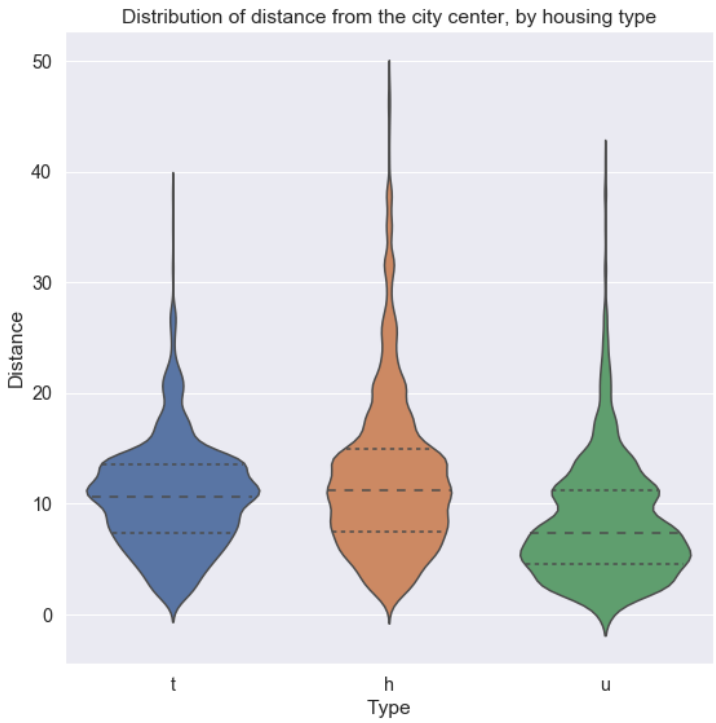
Two features I thought to log-transform were Land Size and Building Area, but chose not to. 2500 observations, mostly of type "unit" (apartment/condominium) were given a Land Size of 0. This is unfortunately not realistic, but the 'Landsize' column proved influential in Principal Components Analysis and helpful in both clustering and decision tree modeling.

Implementation

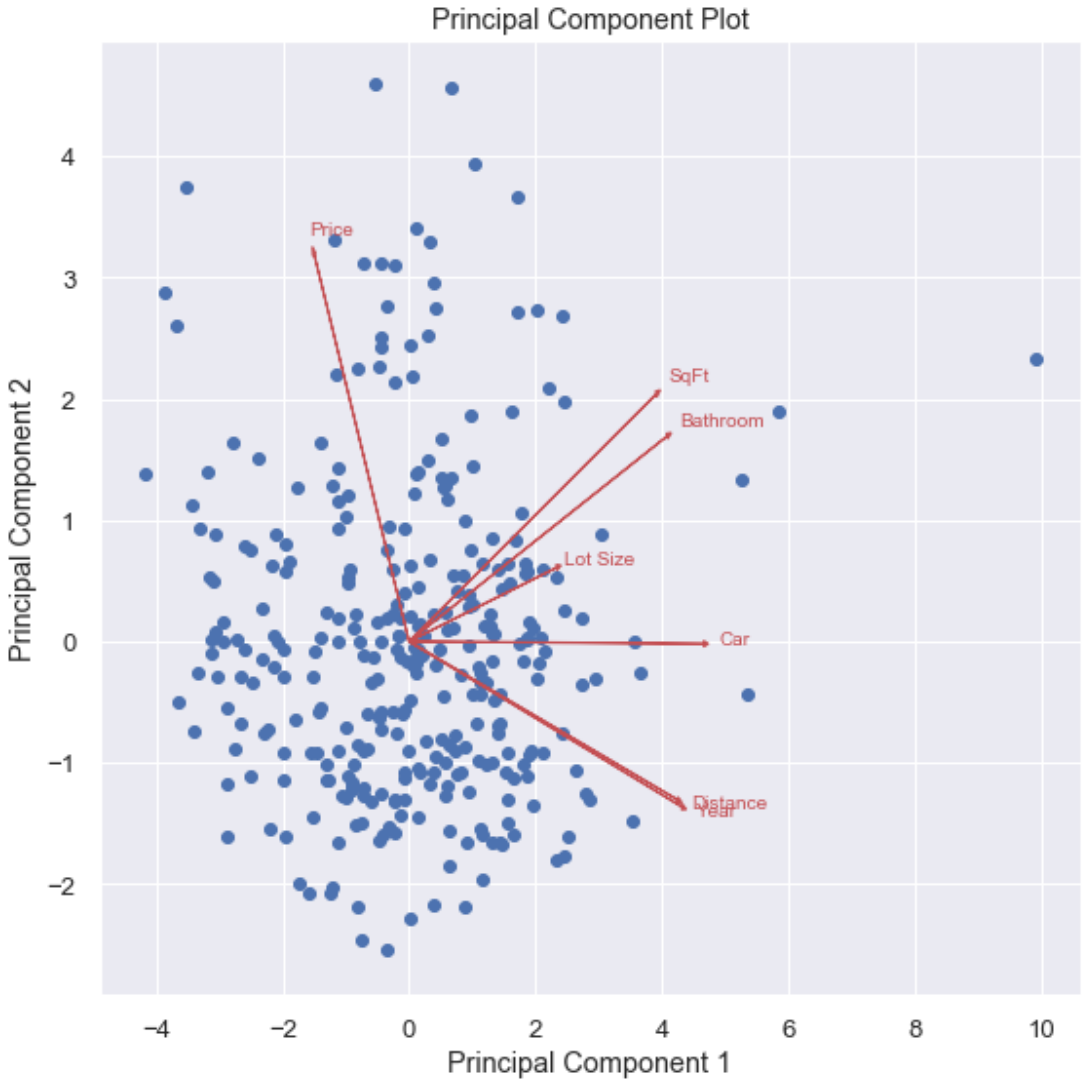
Process:

First I read in the dataset and performed some basic EDA with df.describe(), df.head(), and began looking for outliers or incorrectly-entered values. Typically I do this with multi-feature scatterplots and correlograms.

When I want to know more about a feature's distribution, Seaborn's violinplot() function tells me a lot.

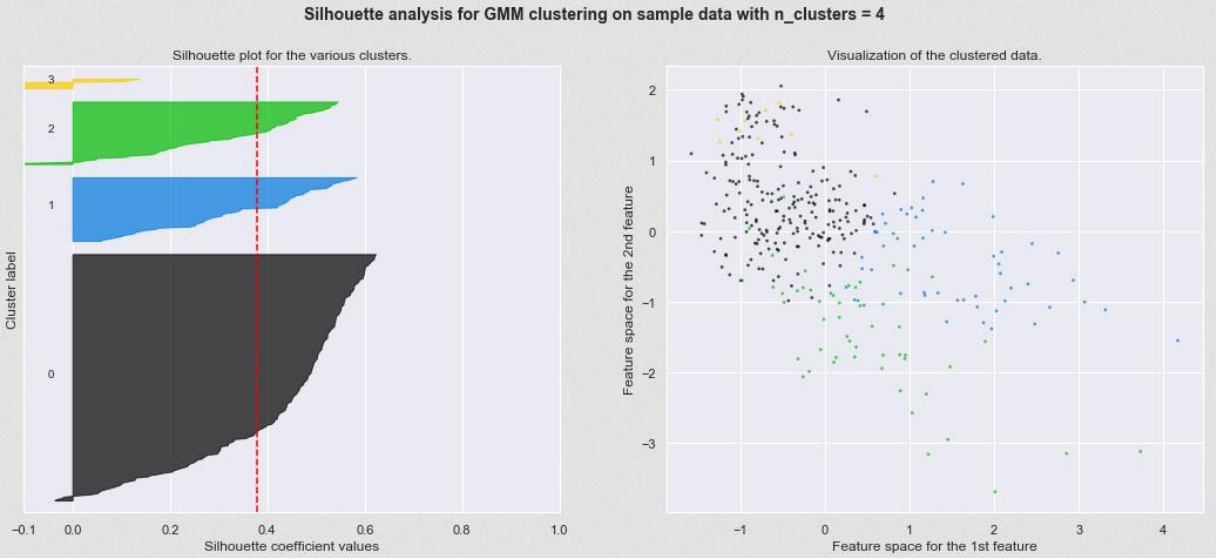


Next, I created a DataFrame of the suburbs, with the aggregated mean values for their 'Price' 'Distance' 'Lot Size' 'Year Built' 'SqFt' 'Car' 'Bathroom'. This gave me about 340 observations. I wanted to understand the relationship between all these variables so I performed principal components analysis on this dataframe by scaling it then using sklearn.decomposition.PCA. Then I graphed feature weights for each principal component and plotted their relationships (in the first 2 dimensions) with a biplot. Obviously a correlogram can show relationships between 2 variables at a time, but I need to see them all together.

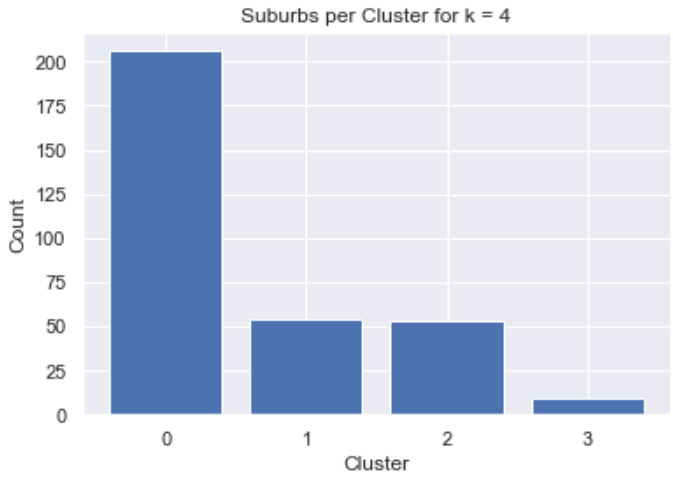


Next came the clustering: I plotted an elbow chart of distortion (the sum of squared distances from each point to its assigned center) left remaining with several numbers of clusters, but I didn't see an "elbow" shape formed. So I used Hierarchical Clustering and measured cluster quality with silhouette score, which measures how similar an observation is to other observations in its cluster. This model was my benchmark.

I was able to improve over the benchmark with Gaussian Mixture Modeling.



Afterwards, I created several graphs and pivot tables to understand my clustering results:



At the end, I was still curious about whether I could predict a house's location just from features of the data, and I wanted to perform an experiment: would KNN Classification be worse than a decision tree at predicting an arbitrary value like location? I used sklearn.neighbor's KNeighborClassifier and ran it with no tuning. Then I used sklearn.tree's DecisionTreeClassifier with no tuning to compare the results.

To increase my predictive accuracy, I used XGBoost's classifier and tuned it with GridSearchCV. GridSearchCV tested my dictionary of parameters:

```
params = {
'max_depth':[4],
'learning_rate':[0.15,0.2],

'gamma':[0.5,0.75,1]}
```

And told me which achieved the best score, when averaged across k=4 folds.

With sklearn's score_report function I could see how well I predicted each suburb:

	precision	recall	f1-score	support
Banyule City Council	0.66	0.51	0.58	157
Bayside City Council	0.79	0.72	0.76	154
Boroondara City Council	0.72	0.79	0.75	369
Brimbank City Council	0.76	0.70	0.73	154

The biggest challenge was keeping track of the proper names for each dataframe I had created. Another challenge was that scikit-learn does not have many built-in graphing functions for its algorithms, so I either had to write my own or find a previous one and adapt it to my needs.

Refinement

To improve my prediction quality, I tuned with GridSearchCV and tested a combination of different parameters (max_depth, learning_rate, gamma). Max_depth controls the number of nodes on the trees. Gamma is the "lagrangian multiplier," and its default value is 0. Conceptually, Gamma is a regularization parameter, it controls the minimum loss reduction required to make a further partition on a leaf node of the tree. Additionally, GridSearchCV has a built-in cross-validation function, which I tuned to 4 folds.

For clustering, I used a function that prints the silhouette score for each of a number of clusters, and graphed the score for each cluster. It's important to see the score for each cluster, because the average value returned at the end may be skewed by one unreasonably high cluster score, while the rest are poor. For help in feature selection, I consulted my PCA biplot to find which features were heavily correlated; adding two strongly-correlated features to the model doesn't add meaningful improvement to your model, and may increase sensitivity to new data (variance).

I achieved a great improvement In silhouette score by using the parameter covariance_type = 'diag' over the default parameter (covariance_type='full').

IV. Results

Model Evaluation and Validation

For Gaussian Mixture Model clustering, I achieved a silhouette score of 38% for 4 clusters. This was an improvement over the Hierarchical benchmark's score of 34.5% for 4 clusters.

Cluster Analysis:

Cluster 1 is overwhelmingly single-family detached housing. It's also the most affordable. Because of its longer distance from the city center, it appears to be semi-rural neighborhoods or what we'd call in America "exurbs". These are smaller, newer homes built on larger lots. They might contain more blue-collar families or young couples early in their career.

Cluster 2 is significantly closer to the city center and older than Cluster 1. It's majority single-family detached and appears to show a much more even mix of housing types, with the largest amount of "townhouse" units. I'm guessing this cluster shows what we call "inner-ring" suburbs in America. This mix appears to comprise the most desirable neighborhoods because of its large average price.

Cluster 3 is the closest to the city center. It contains by far the oldest and smallest units, and exhibits the most even distribution of housing types. No data was available on age of the residents, but I'd assume this cluster contains many college students, young blue-collar workers living together or single, with an occasional luxury high-rise unit, or detached house in-town by the beach. Notably, some of the most expensive units in the dataset were luxury houses close to the city center.

Cluster 4 only has 18 observations which are all single-family detached homes. These exhibit the features of rural, agricultural homes. The average lot size and house size is extremely large. I would consider this, basically, a cluster of outliers that helps keep my other 3 clean and logical.

For Boosted Decision Tree prediction, I achieved an F1 score of 77%. This marked a stark improvement over the K-Nearest Neighbors benchmark of 20.3%. The jump in classification accuracy from a numerical technique to decision tree was to be expected. Conversely, k-nearest neighbors and linear regression provide a better predictor of price (measured using Root Mean Squared Error) than a decision tree, because it's a more logical, less arbitrary value.

Justification

Hierarchical clustering (particularly with ward- or complete-linkage) is a plain good algorithm for this type of problem, making it a tough benchmark. The tree structure is good for arbitrary designations like city borders, and the small number of observations makes the

dataset suited for hierarchical clustering. Across all number of clusters, it maintained respectable silhouette scores (about 30%) if the input features were good. Gaussian Mixture Models with covariance_type ='diag' achieved a silhouette score of 38%, which is useful.

But the justification for moving from hierarchical to GMMs is that we can examine the weight log-probability of an observation belonging to one cluster or another. This is a marked breakthrough in model quality and opens a range of possibilities for application. For example, if a person has a 25% chance of belonging in one cluster of customers, but a 25% chance of belonging to another, a marketer may send them two kinds of ads. If a demographer sees change happening in one type of neighborhood, she now has a probability that the same change will happen in another type of neighborhood.

An increase from 20% to 77% from the benchmark to the final model makes a good justification in classification.

V. Conclusion

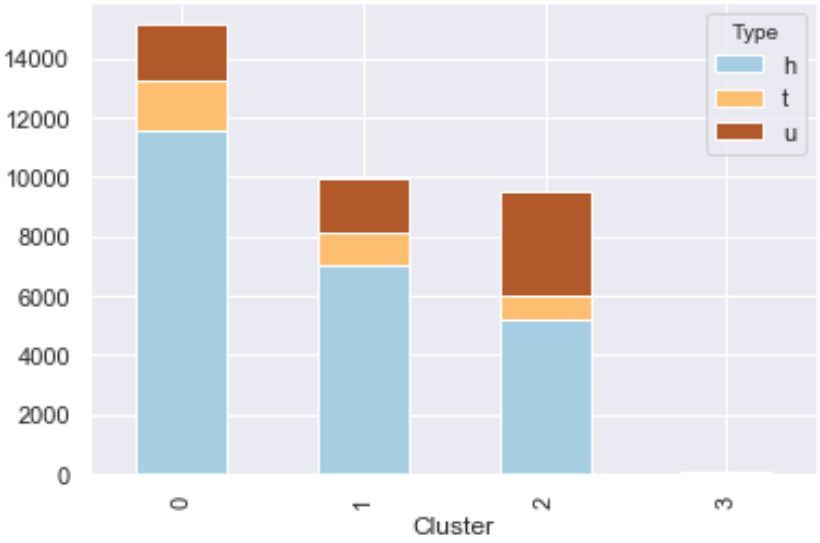
Free-Form Visualization

As shown in this table, Cluster 2 (1 on the chart, because Python is zero-indexed) has the highest average price.

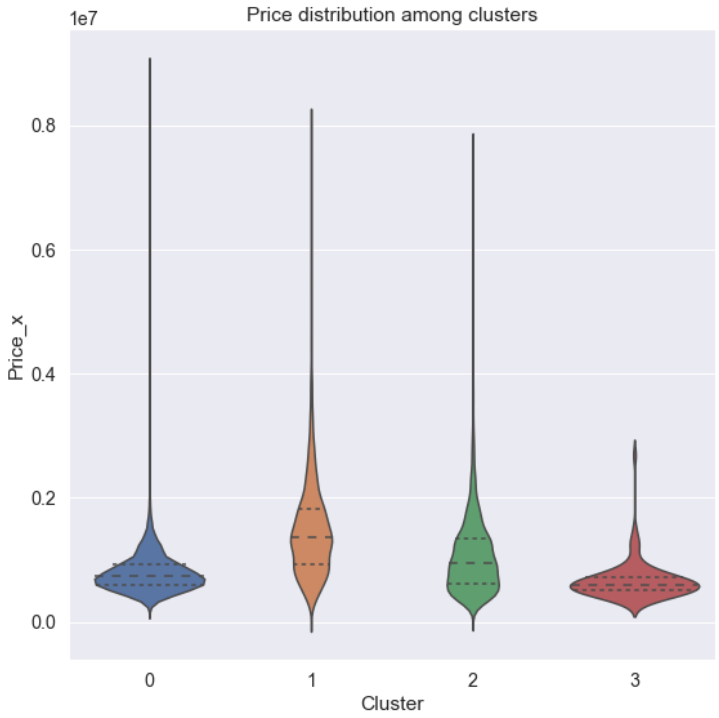
Cluster	Distance	Lot Size	Price	SqFt	Year
0	19.891203	803.068291	7.631998e+05	162.590589	1982.929666
1	11.444527	679.309062	1.458575e+06	194.118203	1964.710908
2	6.189178	779.920932	1.093075e+06	125.804911	1945.214892
3	33.234357	5498.689553	6.882402e+05	179.835802	2003.161300

And, as shown here, Cluster 2 has the highest rate of townhouses (11.3%).

Cluster 1 has the lowest price and the largest percentage of single-family detached housing. Cluster 3 exhibits the biggest mix, but the smallest percentage of townhouses.



Cluster 2 has the largest percentage of townhouses and the highest average price (besides the outlier, Cluster 4). Clusters 2 and 3 exhibit the most variation in price as well. My conclusions are that townhouses appear to drive up a neighborhood's value.



Reflection

It seems that the most valuable communities here have a mixture of apartments, houses and townhouses. It would probably be beneficial to a developer to build a mixture of housing types in a neighborhood. This design would also give residents better access to amenities, and wouldn't need to travel to downtown as much.

In America, it's usually the case that older homes contain fewer bathrooms, but in Australia that's not so. I was surprised to see 'Bathroom' did not show any strong correlation with building age in my graphs above.

Personally, this project taught me the difference between "data science" and "statistics". I understood a lot of the models involved, but reporting on *why* and *how* I did everything made me rethink some methods, and add a few graphs. The most difficult part was keeping track of my dataframes and arrays, what I was doing to them, and making sure I was graphing the right ones. To me, that's an IT problem separate from statistics.

Improvement

Anyone could delete the suburbs with only one observation and instantly see an increase in performance of every model here. I chose to leave them in because I wanted to simulate real-world data that you might find from a municipal site (such as an OpenData program).

Additionally, the Gaussian Mixture Model has a host of interesting visualization and built-in output functions that can relay more information about one's results. I chose not to use these and use silhouette scoring, because these metrics aren't available for other types of models (such as KMeans or Hierarchical Clustering). I wouldn't be able to compare to a benchmark.

If a researcher wants to use scikit-learn's GaussianMixtureModel function, be aware it has Akaike and Bayesian Information Criterion for help in feature selection (built-in as an attribute). My goal was not predicting clusters, just examining them, and I used silhouette score to compare models, so I didn't use AIC or BIC, but for other work it can help.

Python (especially scikit-learn) is limited in its graph capabilities. Occasionally I used the Yellowbrick package to assist in graphing the machine learning. R and MATLAB both have extensive built-in graphing tools that far exceed Python's, and in MATLAB you can easily create 3D graphs. 3D graphs are especially useful for PCA and clustering, because clusters are often not visible in 2 dimensions. Such was the case in my data, where the 2-dimensional graphs didn't show strong clustering, even though silhouette scores told us clusters emerged.