Home Values Prediction

Conner Wattles, Andrew Tingey, Carter Duncan, Niranjan Dawadi, Trevor Wattles
CS 270, Winter 2024
Department of Computer Science
Brigham Young University

Abstract

This project explores the development of a machine learning model to predict housing prices within the Las Vegas real estate market, using data sourced from the Multiple Listing Service (MLS). The study focuses on 11,000 properties sold in the past year, with features such as square footage, number of bedrooms, and geographic coordinates included in the analysis. Data preprocessing involved cleaning missing values, transforming categorical variables, and normalizing features. Three machine learning approaches—multilayer perceptrons (MLPs), k-nearest neighbors (kNN), and KMeans clustering —were implemented and combined into an ensemble model to improve performance. The final ensemble achieved a root mean squared error (RMSE) of $86,099, a significant improvement over the baseline error of $325,883. This work demonstrates the potential of machine learning to provide actionable insights for homebuyers, sellers, and investors, while also highlighting areas for future expansion, such as incorporating geographic proximity to amenities and predicting future home prices.

Approach to Solving Problem:

Predicting housing prices is a complex problem influenced by a multitude of factors ranging from intrinsic property characteristics to broader market dynamics. Our approach systematically addressed this challenge by focusing on data acquisition, preprocessing, model selection, training, and evaluation.

Feature Selection:

To collect the data, we had access to a Multiple Listing Service (MLS) through Andrew's Mom who is a real estate agent in Las Vegas NV. Conveniently, you can filter through homes on the market and homes that have been sold in the entire state of Nevada and export the data as a .csv file. But if you tried to export, say, all data for all homes in the past 10 years, the MLS wouldn't let you because there simply are just too many results. So to start our project we started with just 10 zip codes around where we lived for homes sold in the past 365 days. We decided to not use data for homes currently on the market because that would skew some numbers. Usually the selling price is not the same as the asking price. We exported 8913* zip codes and got over 1800 instances with the following features in our 8913*-sold-last-365.csv:

```
#,Unnamed: 1,Zip Code,Stat,Address,Unit
Number,Current Price,BldgDes,Sub,Approx Liv
Area,Beds Total,Baths Total,Garage,Pv
Pool,Spa,Year Built,Actual Close Date,DOM
```

On our first attempt with this "smaller" data set was to be able to quantize all of the data points. Afterwards, we will download all homes sold in Las Vegas in the past 365 days. However, since this is a much larger area, the MLS forced us to split our data in half and only allowed us to export the ~9,000 instances of homes sold in the past 180 days. To quantize most of the feature sets was very simple because, for example, "Beds Total" is already a positive integer. Others were trivial where "Pv Pool" or "Spa" were just yes/no values mapped to 1 or 0. Some features were dropped for the following reasons:

-"#" was just a counter on the .csv file

-"Unnamed: 1" was actually a unique identifier for this particular MLS.

-"Stat" (meaning status) were all 'S' meaning Sold - we intentionally only included S when exporting, (excluding homes actively on the market)

-"Unit Number": was mostly NaN for all homes since it's in the "Address". And was only included in homes where the

-"BldgDes" was "Condo"

Finally, other features required more creative data cleaning.

-"Sub" (Subproperty type) were mapped accordingly:

```
0: "SFR" (Single Family Residential)
1: "TWH" (Townhouse)
2: "Other" (Condominium / Manufactured Home)
```

Most Instances were either SFR or TWH, so it was rare to see a 2: Other.

We also split the "Actual Close Date" into two separate column values of "Actual Close Month", and "Actual Close Year". In retro-spect, our actual close year most definitely did nothing since it was such a large value and all instances were from the same year.

Impressively we between all 11,000 instances, we only had 2 missing data points. We decided to just drop these two values as well.

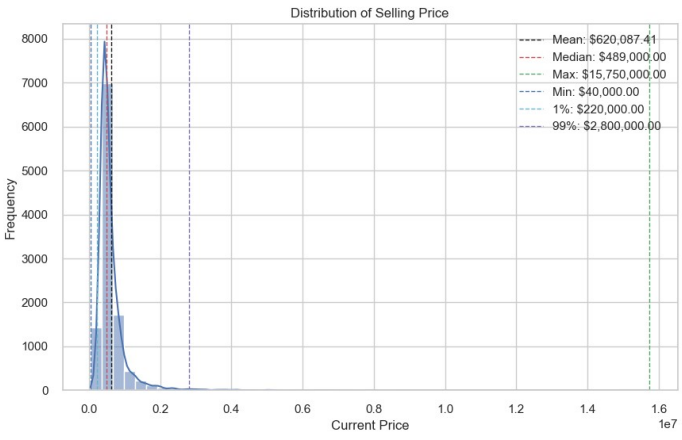
Our output column on this data set is “Current Price”. That is the price that the home sold at.

Initial Results

Our initial model was incredibly bad. We simply plugged in all the quantized values of the data set into a MLP with 100 hidden nodes. We split the data set into 80/20 train/test sets and got a result with a root mean squared error of $289,991. Meaning we were predicting the price of a home, on average, almost 300 thousand dollars off of what it actually sold for. This is only slightly better than our baseline. Our baseline is just guessing the average home price for every home - this gives us a root mean squared error of $325,883. We decided there was a lot we could do to make this worthless model do a lot more. The most important, was to normalize the output data, since we're dealing with home prices from $40k to over $15M. Other ideas we had were to do an ensemble of multiple ML models and to use the homes’ location (latitude/longitude) as another feature to calculate something like distance to school or distance from downtown, etc.

Iterative Steps

We tried many different functions to determine how to normalize our output value of Current Price. Our main criteria was that our normalized values were small, positive, are reversible, and have less of a skew than our current data has. Our current data looks like this:



We first passed our data through a logarithmic function. This helps reduce the large values, and is easy to reverse with just a e^x. We then reduced the data between [0, 1] by passing it through the min-max normalization function. This worked well and we used this function for a while, but our data was still too skewed to the center.

After lots of headache, we finally decided that our data was far too skewed, especially on the higher end. So, we decided to drop all values over the 99th percentile ($2.8M) and below the 1st percentile ($220k). We recognize that this inherently limits our model to some accuracy <98% accuracy, but will definitely help our model in the long run. Again with the logarithmic function, our values have a much smoother distribution. And then with a max-min function, we get our last criteria that our values are between 0 and 1.



Adding Latitude and Longitude

Our other idea was to include latitude and longitude as features of our input data. To do this, we used the

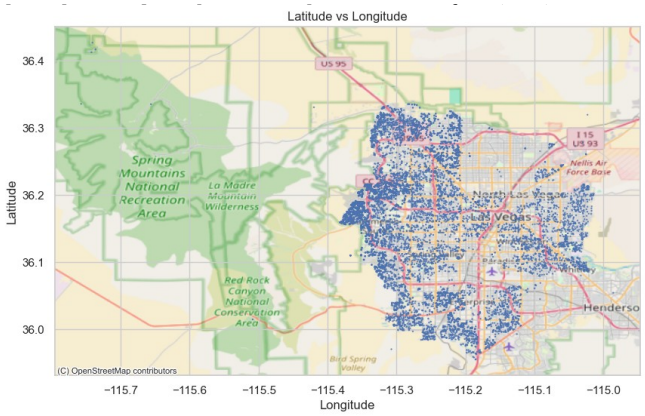


Nominatim library from geopy.geocoders. We chose this one because Google charges you for more than 10,000 api calls a day. This library was free, but took about 1 second per instance. We passed strings to geocode such as this: `f"{address}, Las Vegas, NV, {zipcode}"` But almost 25% of the values returned a location from another state/country or did not return a value. We detected this by filtering latitudes outside the bounds of 33 and 37 and longitudes outside the bounds of -116 and -114. So we put all the failed instances into their own file (~4,000) and began to work without it. With our script working, we were confident to put the last 4000 to the Google geoencoder.

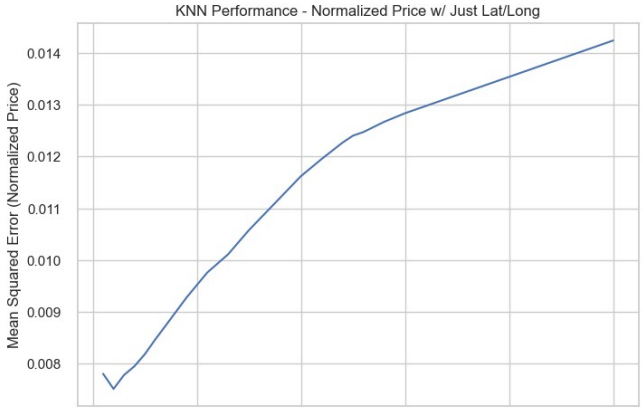
Then we had all 11000 instances geo encoded with latitude and longitude. We decided that that was sufficient headache and calculating distance to nearest school or to downtown would be learned from the model, and not a guaranteed improvement to calculate as an input feature. So we left our two new features as floats [33,37] and [-116, -114]. Plotted on a map it looks like this:

Some things to note about this map:
-There are large blank spaces on the north and south-east sides of Las Vegas. This is because they are technically different cities “North Las Vegas” and “Henderson”. It's unclear if this is from our initial export from the MLS or from how we geo encoded the values. Since almost every instance got a geocoded value (>98%) we assume this is from the MLS. However there could be some error here, since we see about 10-15 points still in these alternate cities.

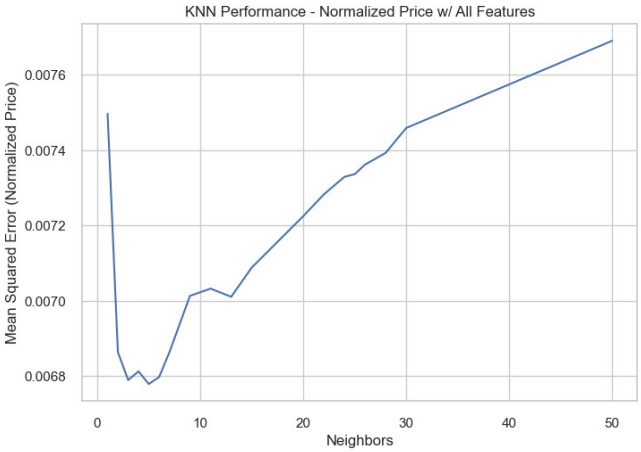
-There are very few homes sold in the mountains (Mount Charleston). The decision could have been made to also



the real world between homes. To determine the optimal number of k neighbors to look at, we ran the model with 10 different random 80/20 train test splits for k neighbors between 1 and 50. We were expecting to have an optimum performance at about 5 or 6 neighbors, but our analysis told us that our optimal kNN is to look at just 2 with minimum average mse around 0.00641. (This number is in weird units since it's from the normalized output [0,1] and is mse instead of root mse, but it will be good to compare to the next model..)

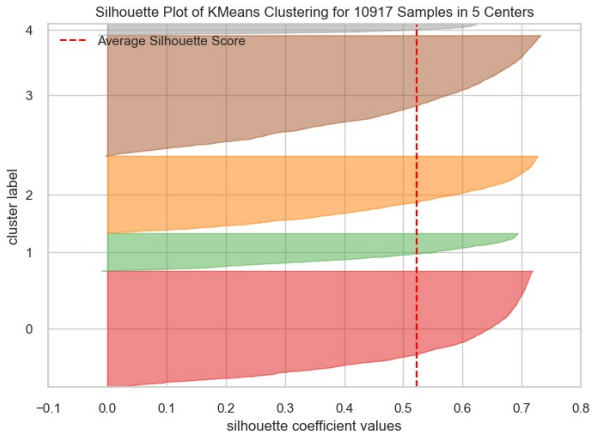


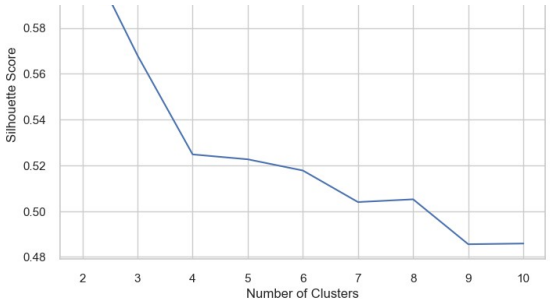
We then were wondering, why limit our input space to just latitude and longitude? Would our kNN model do better if it included all the other input features? Simply including all features shows… not really: our best mse remains around 0.0067, but our best neighbors jumps to 5:



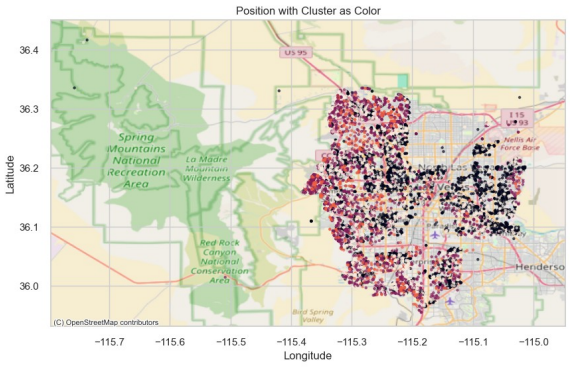
Thus our best kNN model is simply looking at the two closest literal homes’ prices.

KMeans Clustering:
At this point, we knew we wanted to end up with an ensemble of models. And since our kNN model only looks at latitude and longitude, we wanted another model to analyze the rest of the information. A KMeans model could cluster similar types of homes into one group, for another model to predict a home price on. But this requires a set number of k clusters. So we looked at the silhouette plots for all k [2, 10]. A simple eyeball test looks like k = 5 has the most even distribution of 5 different clusters. Also looking at the silhouette score vs number of clusters, it seems like 5 is a good amount of information before we lose too much in clustering collusion.





A similar argument could be made for k = 6, however, looking at the map of each instance's position with clustering as color, the k = 5 gives the best intuition as to how I remember Las Vegas neighborhoods. Where the black points are near inner city, smaller homes and the dark maroon represent the outer city bigger, more expensive homes:



Ensembling all three in a Perceptron:

Our final model includes inputs from the first MLP, the kNN, and the KMeans Clustering models. We still split the data 80/20 again and plugged the values into a perceptron with linear regression to get our final model.

Final Results:

For our final model, we have three simple inputs which are the outputs of the MLP, kNN, and clustering models. We split the data 80/20 again similar to our initial attempt. Training the data is fairly simple since kNN and Clustering models don't need training time. The hyperparameters were selected from the ideas before. The MLP model was trained with its own error loss function, and has an output that is normalized between 0 and 1. After training on the training set, our test set has a root mean squared error (RMSE) of $86,099. This is not as great as we were hoping for, but we were able to decrease our error from our initial attempt. Our baseline idea had an RMSE of over $300,000 and our initial attempt had $289,000 RMSE. So all things considered, our final model is significantly smarter than the first attempt. Although this model isn't as accurate or useful as Zillow's we're still proud of it to be able to minimize the error.

Conclusions/Discussion:

KNN Worked well for This Data

The most influential features we included in our data were the latitude and longitude of the houses. When comparing the mean squared error of all features to the mean squared error for just latitude and longitude, there was only a slight decrease in accuracy which made sense to us. In the real world, houses within the same neighborhood are rather similar in terms of price to one another.

The other features, such as the number of bathrooms, number of bedrooms, square footage, etc. seemed to help fine-tune the model to provide the best results.

Determining K for K-means can be Tricky

The K-Means model for this data proved to be a helpful addition to the ensemble, but determining the K amount of centroids is not a particularly easy process. It takes some exploratory data analysis to figure out the optimal amount that would make future work evermore tedious.

Future work:

Expanding to Other States/Nations/Countries

One aspect to consider expanding on is the variance in our data collection. Currently, our data is limited to houses in Las Vegas, Nevada. This would not be a good model for predicting house prices in New York or Germany, for example.

Additional Features

Some other features that could be worth considering are as follows:

-Distance from schools, malls, entertainment centers, grocery stores, etc.

-Weather/Climate (colder areas may be priced differently than hotter areas)

Data Collection

When it comes to collecting the data, it could be beneficial to set up a web scraper in order to maintain the most up-to-date and current prices for houses in the present-day market. This would remove the need to manually update the data the model uses.

If this is not done, the model will lose its accuracy over time because the data it was trained on is outdated. The challenge with this approach is ensuring the same measurements are being used, especially if the web scraper looks at several different websites with different measurements.

Predicting Future Prices

It could be interesting to modify our model to predict future prices of current houses. This would be beneficial in determining if a home has the potential to increase in value from the initial price or if it will decline in the years to come. Pivoting from current home prices, the model could be adjusted to predict prices for new developments. For example, if there exists an area of land that could be turned into a housing complex, it could be beneficial for housing contractors to have a machine-learning model that predicts what quality and style of housing would be the best to build.

Predicting Rental Prices

Our model only predicts the price of houses. It would be interesting to incorporate support for predicting apartment rental prices. This can give apartment complexes an idea of what is a fair price to charge for rent/parking given the amenities and overall quality of the rooms.

Another worthwhile addition would be to predict Airbnb rental prices. This can be beneficial for both the renter and the owner. The renter is less prone to overpaying and the owner is less prone to undercharging.