CS260: Project Report

Daniel Geng
Fall 2015

Abstract

This project examines Manhattan real estate prices and its correlation with zip code (which generally determines its neighborhood), square feet, number of bedrooms/bathrooms, and year of construction. Many models were used to predict prices with varying performance, including linear regression, decision tree regression, stochastic gradient descent, KNN regression, and neural networks. Data was restricted to only downtown Manhattan (below 30th street), due to both investor appeal and to prevent over-complexity of the zip code feature. While some of the models had acceptable performance, none of them made extremely accurate ($<$10% error) predictions. Because there were only 5 relevant features that were easily extracted, many samples had similar features despite having different prices, making it difficult for the models to be precise. Also, because the training data was from sold listings (2004-2015) and the testing data was from current listings, it was difficult to adjust for inflation. The median house prices in NYC increased by around 26% from 2006-2015 (approximately 3%/year). [1] As a rough estimate, training labels were multiplied by 1.32, which increased the performance of all models.

1 Motivation

I studied Manhattan real estate because it is currently an appealing investment opportunity, with prices steadily increasing year after year. The scope of the project was narrowed to only downtown Manhattan, as it is the most consistent for investments; most neighborhoods are relatively safe and almost all neighborhoods have increasing price trends. The ongoing gentrification of neighborhoods such as the Lower East Side and Alphabet City will undoubtedly increase prices in the next few years.

Although midtown Manhattan also has potential investment opportunities, there are some neighborhoods that currently have high crime rates, which may share zip codes with expensive properties, creating a lot of variance in price among properties in the same zip code. With limited space on the island, it is inevitable that the entirety of Manhattan will eventually be priced too high for average people. While Manhattan is still financially accessible in many neighborhoods for the majority of people, investing in its real estate is an exceptional choice for future-proofing one's finances.

Creating a model that could learn how much property should cost based on prior prices would potentially benefit both investors and property owners. There are many features that are ubiquitous among properties, such as square feet, number of bedrooms and bathrooms, neighborhood, distance to landmarks and/or subway stations, year of construction. A model that could use simple information that any property in Manhattan should have documented would give an objective price which buyers and sellers could negotiate around. Of course, there are many features that are either impossible to quantify or difficult to obtain which may be important to people, including building history, amenities, pet policy, quality of nearby services, ISP availability, interior quality (i.e. remodeling, etc.), type of neighbors, etc.

The importance of subjective features can be determined case-by-case. A predictive pricing model would simply prevent users from paying too much for a property or pricing their own property too

low. As other parts of New York City become appealing investment opportunities in the future, there will be motivation to either augment the existing learning model or train/develop a new one. For example, if some neighborhoods in Brooklyn or Queens start to have high price growth, a new algorithm will probably have to be developed; a feature such as distance from nearest subway station will be weighted very differently for Manhattan and other boroughs.

2 Background

While there are websites that have APIs which can list aggregated data such as median prices and square feet by neighborhood, there is no such option for viewing property data that has specific features. Websites such as `www.streeteasy.com` and `www.trulia.com` make it possible to view individual listings which can be narrowed down to very specific criteria, but there is currently no method for a "reverse search" that allows both property owners and prospective investors to determine how much they should be pricing or paying for a property with certain specific features. While it is possible to gain some idea on how much a property should cost through manual brute-force searching on real estate websites, having a model that allows simple input of features would greatly decrease the burden of research.

I have some personal experience with Manhattan real estate from my three years of living there. I worked closely with a real estate broker and visited over 100 properties during my time there. However, I have not done any formal research or computational analysis on the subject. The intuition that I have regarding how much a property should be priced at is derived purely from my own experiences.

3 Methods

The median house prices in New York City increased by around 26% from 2006-2015 (approximately 3%/year). [1] As a rough estimate, training labels were multiplied by 1.32, which significantly increased the performance of all models (as measured by percent error). While this metric is obtained from the entirety of NYC, I felt that this trend was a close enough estimate of current prices. Furthermore, predicted values were plotted against actual values to determine how outliers were evaluated in each learning model.

3.1 Neural Networks

Originally, neural networks were used as the primary learning model. Neural networks seemed appropriate for this project because there can potentially be a large number of features, which may or may not be known. Also, because weights between neurons are updated based on additional training data, a predictive price model based on a neural network could be updated with new data to keep up with CPI (consumer price index) inflation and housing price increases. SLSQP (Sequential Least Squares Programming) was used for the optimization of the cost function.

Features were normalized by dividing by the maximum value of each training feature. Both the sigmoid and $tanh$ activation functions were used. The number of nodes in the hidden layer and lambda value were adjusted to many values, but good performance could not be obtained. Various combinations of input features were omitted in an attempt to diagnose the poor performance. However, nothing significant was being learned, so other learning models were introduced to the problem.

3.2 Regression

Various forms of regression were used in an attempt to achieve greater performance, including KNN regression, linear regression, lasso regression (L_1 norm), ridge regression (L_2 norm), decision tree regression, and stochastic gradient descent regression. *1-to-n* encoding was used for encoding the zip code input feature. There were a total of 17 zip codes in the aggregate data used; the zip codes with under 100 training samples were either mislabeled (which was fixed by manually searching for the address online) or removed from the data set. Other methods for using the zip code as a feature were also tested, such as converting each zip code to a ratio of its median historical prices. For example, the zip code with the greatest median price (10013) would be scaled to 1.

3.2.1 Nearest Neighbors

Two distance metrics were used for KNN: Minkowski and Euclidian (L_2). For prediction, two weight functions were used: uniform weights among all points in each neighborhood and distance weights which gave higher influence to points that were closer. The hyperparameter k was calibrated by iterating from $k = 3$ to $k = 80$. The value of k with the least overall error percentage was used as the model's performance measurement.

A slight variation to KNN, radius neighbors, was used as well. This algorithm used a fixed radius to determine the points that belong to a sample's neighborhood. Theoretically, with a small radius, radius neighbors should achieve similar performance as KNN. With a larger radius, points that are closer could be assigned greater weights.

3.2.2 Linear

The first model used was ordinary least-squares linear regression. Then, a linear model was trained using the L_1 norm for regularization (Lasso). Various alpha values (between 10^{-7} and 1) were used and the "random seed" value was set to 0 so that results could be consistently reproduced.

Ridge regression (L_2 regularization) was used with the same alpha values. Again, the random seed was static so that results could be consistently reproduced.

3.2.3 Decision Tree

Various hyperparameters were calibrated, including maximum features used to consider the best split, minimum leaf size, minimum split size, maximum depth, and maximum leaf nodes.

The Gini importance for each feature was used to gain insight on which features are the most relevant to price prediction.

3.2.4 Stochastic Gradient Descent

Least-squares was used as the loss function. Different penalty calculations were used, including L_2, L_1, and elastic net, which linearly combines the L_1 and L_2 penalties. Various alpha values (between 10^{-7} and 1) and learning rates η (between 10^{-5} and 0.5) were used.

3.3 Normalization

Two methods were used to normalize the input features. The standard scaling method of subtracting the training mean and scaling to unit variance on each numerical feature was used. In addition, because there were a lot of outlier properties with exorbitant prices, a "robust" scaling method was used as well, which subtracted the median and scaled the data according to the interquartile range (25^{th} - 75^{th} percentile). Each algorithm used both scaling methods with marginally varying performance.

4 Results

As mentioned before, the performance using neural networks was quite subpar, even after testing using many different combinations of hyperparameter values. Typically, there was greater than 500% error. However, there were some good results with some of the regression methods used ($<$30% error). For all models, 1-to-n encoding of the zip code feature was much better than using the ratio of the medians. Also, this allowed the models to learn the importance of each features on its own; using the ratio of medians introduces some outside bias and greatly affects the model.

The best performance was achieved using KNN. Initially, with standard normalization, there was 28.046% error (k=12); with robust normalization, there was 27.880% error (k=11). With the inflation factor added, error dropped to 20.237% (k=12) using standard normalization and 19.842% (k=14) using robust normalization. Intuitively, robust scaling on the features makes more sense because of the presence of outliers for square feet, number of bedrooms, and number of bathrooms.

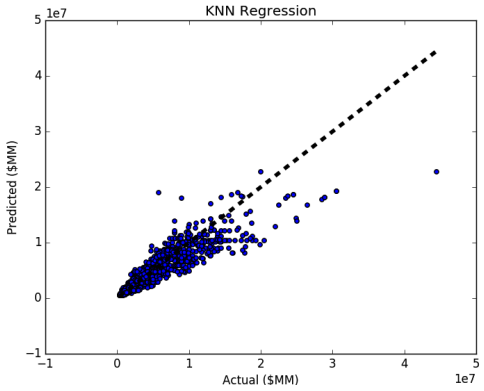
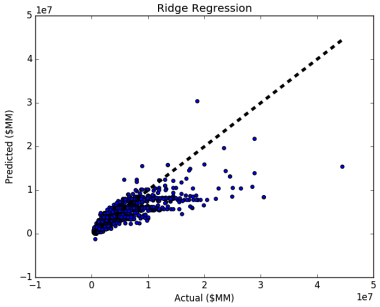


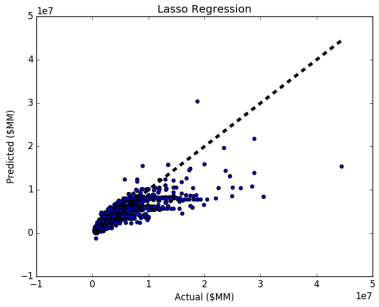
Figure 1: Test data, robust normalization, k = 14

From the graph, it can be seen that the majority of prices are predicted relatively close to the actual listing prices. The outlier farthest on the right is a mansion with 11,300 square feet, 6 bedrooms, and 7 bathrooms that sold for $44.5MM. Because its 14 nearest neighbors had an average of approximately $22.5MM, the prediction had approximately 100% error. With a lower k hyperparameter, this sample would be classified closer to its actual value because there were similar samples in the training data, but most of the other samples would have greater error.

Regular linear regression, lasso regression, and ridge regression had worse performance than KNN, but had similar results compared to each other, with 32.346% error, 32.341% error, and 32.342% error respectively.



(a) Test data, ridge regression, $\alpha = 1$



(b) Test data, lasso regression, $\alpha = 1$

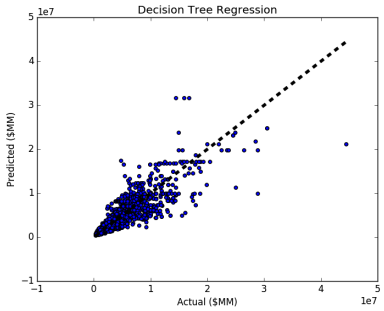
As expected, the graphs for lasso and ridge regression have very similar geometry.

Again, there was a large error with the highly priced outlier sample. The predictions generated from all three models were very similar, usually within 0.1% with each other. Typically, they all under-predicted the prices, but there were a few samples where there was very high over-prediction (i.e. $2MM prediction for $720K actual). There were some strange predictions on the other end as well, with some properties predicted as being priced under $100K. This is likely because a linear model does not have the required complexity for the features provided, especially due to outliers.

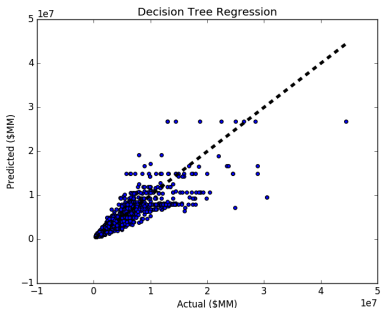
Decision tree regression had better performance than the linear models, with an error of 21.799%. The greatest performance was achieved using $log_2(21) = 4$ features and 5 minimum samples per leaf. Changing the other hyperparameters did not positively impact the performance, meaning the maximum depth and maximum leaf nodes were infinite and the minimum samples for splitting a node was 2.

There were some interesting results from observing the Gini importance index of each feature. Square feet had a Gini importance of 0.61016, which was much greater than any other feature. This made sense intuitively because larger properties should have higher prices. Bedrooms had a 0.13488 importance and bathrooms had a 0.18742 importance, which showed that additional bath-

rooms have a greater correlation with price. Prewar had an importance of 0.00057, which shows that prewar status is not very important in determining price. The feature associated with the zip code 10013 had an importance of 0.0633, which was much greater than any other zip code, except for 10280 with an importance of 0.0383; all other zip codes had importance less than 0.03.



(a) No minimum leaf size (25.096% error)



(b) Minimum leaf size = 5 (21.799% error)

From the graph, it can be seen that changing the minimum leaf size hyperparameter caused some of the samples to be predicted with the exact same price, which contributed to reducing the errors caused by outliers.

Stochastic gradient descent regression had an error of 28.038%, using L_2 loss, $\eta = 0.01$, and $\alpha = 0.0001$.

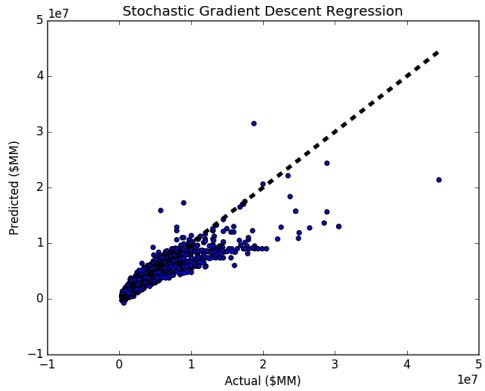


Figure 2: L_2 loss, $\eta = 0.01$, $\alpha = 0.0001$

Like the linear models, stochastic gradient descent under-predicted most of the prices.

Because outliers greatly affected the performance, some of the outliers were removed from the testing data. Just by removing the one property priced at $44.5MM, the error for KNN was reduced by 0.025%. After removing all properties priced over $20MM, error dropped a total of 0.12% to 19.721%.

To retrain the model, the same procedure was applied to the training data. The highest priced training sample was removed ($49.5MM) and the error dropped by another 0.02%. After all of the properties priced over $20MM were removed from the training data, the error was reduced to 19.650% (k = 15). However, even with some of the outliers removed, the robust scaling method had better performance than standard scaling (1% difference).

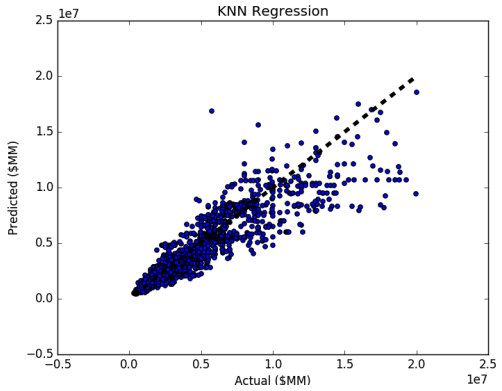


Figure 3: Price over $20MM removed, k = 15

Similarly, with the outliers removed, the other models reduced in error by an average of 0.5%.

Even though the regression models had acceptable performance and produced reasonable predictions, I was disappointed that the neural networks model was not able to achieve similar performance.

However, with the outliers removed, the neural network model was able to achieve much better performance than before. Initially, it had over 500% error regardless of hyperparameter values, but this was reduced to 57.152% using a hidden layer size of 15 nodes and $\lambda = 0.0001$. While this is still significantly worse than all of the other models used, it shows the profound effect that a few outliers can have (less than 0.5% of the initial data in each set).

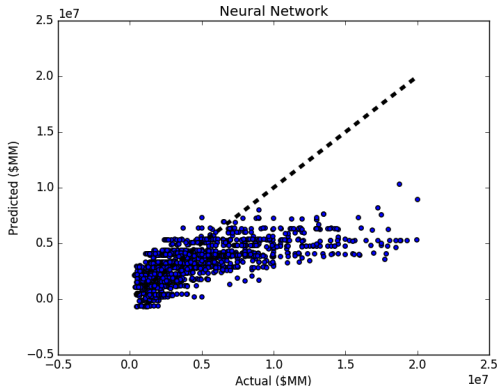


Figure 4: Price over $20MM removed, $\lambda = 0.0001$

The biggest issue with my neural network is that it predicted negative prices for some of the samples. This can be fixed in some way to improve the average percent error. For example, the negative predictions could be set to some minimum value. However, this does not represent the function of a neural network, so I did not modify any of the prediction prices.

Although some of the models were able to reasonably predict prices, I would probably not use any of the models if I decided to invest in downtown Manhattan real estate. I feel that the lack of viable features adversely affected the performance, as there are many more factors that can be used to predict price that were unfeasible to use, such as distance to nearest subway station or specific landmarks, crime rate, and education level.

5 Data

All data was obtained from the NYC real estate and rental website `www.streeteasy.com`. The property listings that were already sold were used as training data (19,355 samples after filtering) and the current listings on the market were used as testing data (1,997 samples after filtering). While it can be argued that listing prices are greater than actual sale prices, this is not necessarily true in downtown Manhattan. Often, especially in "trendy" neighborhoods, there may be a lot of prospective buyers for a property and the sale price could exceed the sale price. Of course, the inverse happens as well. Due to complexity, these factors were not considered for scaling training or test prices.

Because the StreetEasy API was only able to provide aggregated results based on specific search queries, a simple Ruby program was written to fetch search results from various queries. The individual listings were parsed and written into .csv files. The listings from the following neighborhoods were used: Battery Park City, Chelsea, East Village, Financial District, Flatiron, Gramercy Park, Greenwich Village, Lower East Side, SoHo, TriBeCa, and West Village. Because of low sample sizes ($<$50), Chinatown, Civic Center, Little Italy, NoLIta, and Stuyvesant Town neighborhoods were excluded. Neighborhoods were represented using their respective zip codes to reduce ambiguity, as there are no official boundaries for any neighborhood.

While a neural network should be able to work even if some features missing, only some of the information that I intended to use was available from the search listings page (only five features). Therefore, I made sure that all samples had non-null values for zip code, square feet, number of bedrooms and bathrooms, and whether a building was constructed prewar. Prewar refers to World War 2; buildings that finished construction 1940 and earlier are considered to be prewar. [2] While some residents prefer the history, limited availability, and architectural style of prewar buildings, others prefer the modern amenities and general reliability of newer constructions. Thus, I thought it would be interesting to see if prewar construction status had any significant correlation with price.

Some samples, from both the training set and test set, were removed because they were anomalies within the overarching data sets and would not benefit the learning models. For example, recorded sales of entire buildings were part of the data, selling for $80MM with 165,000 square feet, but 0 bedrooms or bathrooms. Also, there were a few listings that had over 1,000 square feet but 0 bedrooms (studio apartments) that sold for over $1MM. However, after doing some research, these listings were just abnormally spacious studios with unique amenities such as an indoor gym/pool. These were not removed from the data sets. A sample count of each zip code was mapped and there were a few zip codes containing under 5 listings (likely due to typos when listed). These were fixed manually by searching for the building itself on Google Maps.

6 Software Tools

Ruby was used for collecting and parsing the data from StreetEasy. The *open-uri* and *json* gems were used.

Python was used for training models, predicting prices, initial data analysis (i.e. removing bad samples), and graph plotting. The **scikit-learn** library was used for most of the regression models and data preprocessing. The **numpy** library was used for matrix operations and mathematical calculations. **scipy.optimize** was used for minimizing the cost function for the neural network model. **matplotlib** was used for creating plots. The *csv* library was used to read .csv files.

References

[1] "New York NY Home Prices & Home Values — Zillow." Zillow. http://www.zillow.com/new-york-ny/home-values/

[2] Mooney, Jake. "Postwar, Prewar and Everything Before." The New York Times. The New York Times, 03 Nov. 2012.