Zestimate Bazinga: Predicting the Selling Price for Condos in Downtown Vancouver

Andrey Koch[1], Marina K Peremyslova[2], and Lucas Lemanowicz[3]

***Abstract*— Predicting the price of a home is part art, part science. This paper applies linear regression, neural networks, random forest and gradient boosted trees to look at how well these models can be used in predicting the selling prices of condos in downtown Vancouver, Canada. Gradient boosting (GB) shows the best performance, with random forest and neural networks in close second place. Our best method (GB) gives test set $MSE = 0.09$ and $R^2 = 0.91$ with reasonable generalization $MSE_{CV}/MSE_{train} = 1.25$.**

I. INTRODUCTION

The real estate market in large metropolitan areas across USA and Canada is characterized by high volatility. Home prices in popular technological and cultural centers, such as Vancouver, Canada, have been reportedly growing over the last decade owing to a constant influx of people, including immigrants, attracted by a combination of career opportunities, and a superb geographical setting by the ocean bay and nearby mountains. As a result, predicting home prices has become a big challenge. Real estate agents use their domain knowledge to estimate a home price aiding sellers and buyers in the transaction. This estimate is often very subjective and facilitates bubbling the home prices, especially in highly attractive areas like Vancouver. Therefore, our main study goal was to come up with an automated way of price-tagging a home based on its characteristics including floor area, number of rooms, location and others.

The input to our algorithm is a dataset of all condo listings under $2.5MM CAD in downtown Vancouver between January 2016 and October 2018, containing approximately 50 features after pre-processing. We then use linear regression, neural networks, and boosted tree models to predict the expected selling price of a condo.

II. RELATED WORK

There is a good number of articles related to real estate pricing predictions. In general, it is difficult to compare the results given the diversity of features used to model the predictions and relevant error analysis. However, there are some common themes that we tried to reproduce and improve upon in our research. More often than not, the authors attempt to use linear regression and boosted trees regression algorithms [6]. In this example, the R^2 received were 0.73 and 0.9194 respectively. However, no further error analysis was done, so no conclusions about model variance and bias can be inferred.

[1] andkoch at stanford.edu
[2] mpkoch at stanford.edu
[3] llemanowicz at stanford.edu

Another author [7] states that they were able to create a model that explained 80% of the Boston housing sale price variance by simply using the neighbourhood and total square footage features. Expanding the feature set to 36 (unnamed) features boosted the R^2 to 0.92. However, no specific details of the latter model were given.

In another example related to Bay Area house pricing prediction [8] , the author worked with a data set-up similar to the one we were able to obtain, i.e. the initial data set contained the information about various house features, zoning etc. Their best result was achieved on 19 features with gradient boosting (R^2 of about 0.6616), although the information about houses location and surroundings were not used in the final model. In our opinion, those pricing factors are important and should not have been overlooked. So we used this article accuracy results as our benchmark and tried to improve upon it.

III. DATASET & FEATURES

A. Raw Data

The original dataset we received had exhaustive information about all condos listed for sale under $2.5MM CAD in downtown Vancouver between January 2016 and October 2018. The data was pulled from an official Canadian real estate listings database called MLS with the help of a local real estate agent, and contained approximately 10,000 listings. Each listing had up to 237 features including immanent property characteristics like square footage, number of bedrooms and bathrooms, maintenance fees, and relational characteristics like address, vicinity to schools and public transportation, and views. The features can be classified into three categories: structured data (e.g., total floor area); semi-structured data (e.g., address); unstructured data (e.g., listing agent comments). Furthermore, data can be categorized into various types: interval-scaled variables (e.g., number of bedrooms, year built, etc.); temporal (e.g., date property was sold); rank (e.g., floor number); boolean (e.g., fireplace yes/no); categorical (e.g., dwelling type).

B. Feature Engineering: Geocoding and Bucketing

Our dataset did not have the geographical location of homes originally, only the physical addresses. Since location is supposedly important for home value, we used the Google Maps API [12] to geocode condo addresses to geographical coordinates (latitude, longitude). In addition, since the area of interest was relatively small (only about 9 km^2), we approximated it with a flat rectangle and converted geographic to Cartesian coordinates mapping all condos to a

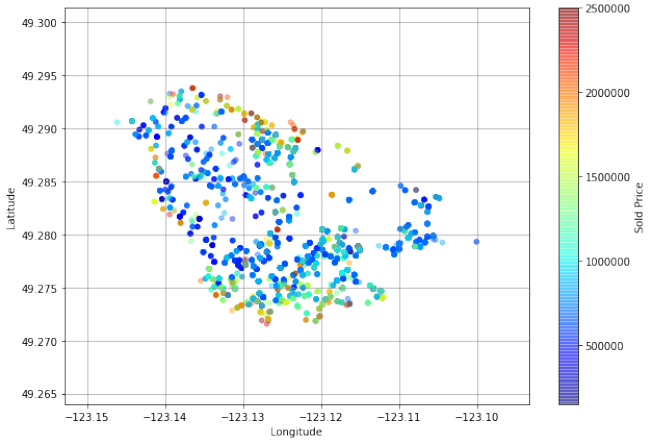

Fig. 1: Condos sale price distribution on the map

rectangular grid of 35 square districts (500m x 500m each) to approximate neighborhoods. The district variable was one-hot encoded and included in the final dataset.

C. Feature Engineering: View Scoring

Figure 1 shows the selling price of condos plotted against their geographic locations. One can empirically see that properties adjacent to the edges of research area (along the waterfront) tended to be more expensive. This suggests that the view is an important feature in determining the price of a condo. Therefore, we created a new feature called the "View Score" by mapping the free-text "View Description" field onto an interval scale. We formed a dictionary mapping all of the words from this field to more broad view categories such as: water, mountains, city park, and urban. For example, the "water" category included keywords like "sea", "marina", "english" (referring to English Bay), etc.; the "urban" category included "church", "stadium", "cityscape", etc. We also included abbreviated words (e.g., "hrbr" – "harbor") and misspelled words (e.g., "poark" – "park"). We added two more categories characterizing the quality of the view: panoramic and partial. The keywords for the first category included "panoramic", "outstanding", "180" (meaning 180 view), etc., and the keywords for the second category included: "partial", "peek", "peekaboo", etc. We assigned scores for view descriptions containing words in any of the categories. View descriptions involving "water" and "mountains" received 2.0 points, "city park" received 1.5 points, "urban" received 1.0 points. Accumulated scores were increased or decreased by 50% if the property had words indicative of the view quality (panoramic or partial, respectively).

D. Data Cleaning

Cleaning the data was essential to having an accurate model, since it was originally input by real estate agents and was prone to mistakes. We removed outliers for the numerical features using three sigma rule. We then imputed data for features that had occasionally missing data (less than 1%) with medians. Lastly, we standardized the data. The final feature set consisted of 48 features.

IV. METHODS

A. Error Analysis

To improve the quality of our predictions we performed error analysis with k-fold cross-validation (CV). We split our dataset randomly into a training and test set (80%/20%). The training set was used in CV-setting where it was sequentially split into training and validation subsets k=5 times. The training subset was used to fit the model to the data, and validation subset was used to compute errors. The average of k errors, the CV-error, characterized how accurately our model performed.

We used two main metrics for calculating errors; mean squared error:

$$MSE = \sum_i \frac{(y_i - f(x_i))^2}{n}$$

and coefficient of determination (R-squared):

$$R^2 = 1 - \sum_i \frac{(y_i - f(x_i))^2}{(y_i - \bar{y})^2}$$

with y_i = predicted variable observations, $\bar{y}$ = its mean, x_i = vector of independent variables (features), $f(x)$ = the model mapping features x on y, and n = number of observations ($i = 1, .., n$). MSE characterizes the average of squared deviations of predictions from observations with MSE = 0 corresponding to an idealistic model exactly mimicking observations. R^2 measures how well the model captures variability of observations given observed features with $R^2 = 1$ being an idealistic scenario. One can show that these properties are closely related such that

$$R^2 = 1 - \frac{MSE}{Var(y)} = \frac{Var(y) - MSE}{Var(y)}$$

giving it the meaning of the fraction of explained variance in y.

We calculated MSE and R^2 for both CV (as explained above) and training sets and also calculated the ratio of MSE on CV and training sets to characterize how well our model generalized to new data (model variance): $Var = MSE_{CV}/MSE_{train}$. A good model would have this metric not much larger than 1. A $Var \gg 1$ would mean we overfitted the data, and our model would most likely perform badly on new data. Finally, after tuning each model and obtaining best set of coefficients, we calculated MSE and R^2 for test set as the final unbiased accuracy characteristic.

B. Linear Regression

We used multiple (multiple predictors) linear regression as our benchmark model. As the name suggests, it assumes a linear relationship between the features and the predicted (target) variable, and treats it as a linear combination of features: $f(x) = \theta^T x + b$, where x is the feature vector, θ is the vector of model coefficients, and b is the bias. To train the linear regression model, one needs to find the

coefficients θ given data X and target variable observations y as approximation of the predictions of $f(X)$. By minimizing the least squares cost function:

$$J(\theta) = \frac{1}{2}\sum_i (f(x_i) - y_i)^2$$

w.r.t. coefficients θ, they are found effectively using normal equation:

$$\theta = (X^T X)^{-1} X^T y$$

C. Regularization with LASSO

As with any method that uses features on different and non-comparable unit scales, the problem with linear regression is that features having larger units tend to contribute to the final result more than features with smaller units, even if their dimensionless variances are comparable. This is mitigated by adding regularization term to the cost function such that coefficients for larger scaled features get penalized. LASSO is a popular regularization technique that uses the L1 norm:

$$\theta = argmin_\theta J(x) + \lambda \sum_j |\theta_j|$$

The added benefit of LASSO regularization is that it sets coefficients of unimportant features to 0 and can be used as a feature selection technique for other methods. This was precisely the reason we used LASSO in our research.

D. Neural Networks

One property of neural networks that makes them a popular ML method is their ability to perform end-to-end learning: given some input features x, a network is able to determine the appropriate intermediary features and weights of those features on its own [11] (unlike a linear model). In the case of predicting the price of a condo for example, these intermediary features could be the neighborhood quality or family-friendliness. This ability makes it a powerful choice for modeling the selling price of condos.

A neural network's ability to model non-linear data stems from its use of activation functions in between its neuron layers. One example of a commonly used activation function is the Rectified Linear Unit (ReLU) function:

$$ReLU(x) = max(0, x)$$

Its main advantage is that it has a very simple gradient and doesn't suffer from vanishing gradients at extreme values, although it can cause "dead neurons" when the product of the weights and inputs skews negative.

The Leaky ReLU activation function addresses this shortcoming:

$$LeakyReLU(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{otherwise} \end{cases}$$

where α is a small number (e.g., 0.1).

Dropout layers are commonly used to address overfitting in neural networks. This is implemented by randomly dropping a small percentage of nodes in a layer during each update iteration, preventing the network from over-relying on any individual neuron. Neural networks learn through the backpropagation of error gradients, and the weights w_l at a layer l are updated by:

$$w_l := w_l - \alpha \frac{\partial J}{\partial w_l}$$

where α is the learning rate, $J = \frac{1}{m}\sum_{i=1}^{m} L_i$ is the cost function, and L_i is a loss function (least squares in our case) for an i^{th} example.

E. Random Forests and Gradient Boosting

As alternate methods accounting for nonlinear relationship between features and target variable, we use tree-based techniques, Random Forest and Gradient Boosting. The random forest (RF) is a special case of ensemble ML methods when individual models are combined together to produce balanced model that have higher generalization power than separate models.

In the case of RF, individual learners are regression trees termed as:

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m)$$

where feature space is partitioned into M regions $R_1, ..., R_M$ and the $c_m = ave(y_i | x_i \in R_m)$. At each tree node, the binary partition is performed on one variable. Finding best binary partition in terms of split variable j and split point s by minimizing sum of squares is computationally infeasible, and a greedy algorithm is used when decision is being made only one step forward. Starting with all the data, a pair of half-planes is defined:

$$R_1(j, s) = \{X | X_j \leq s\}$$

$$R_2(j, s) = \{X | X_j > s\}$$

Then j and s are found by solving:

$$min_{j,s}[min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + min_{c_1} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2)]$$

After finding the split, the data is partitioned into two regions and the splitting process is repeated on both regions and all subsequent regions until some stopping criterion is met. Among different criteria, most popular is stopping growing a tree when minimum node size is reached. Individual trees are prone to overfitting, and RF method overcomes this problem by combining multiple trees grown on separate data subsets. The default approach to forming subsets in RF is bootstrap sampling with replacement when dataset size remains the same but its composition varies among samples. This way, overfitting is decreased as each individual tree is learning from a different subset of data. Moreover, a random subset k rather than the whole list of features m is considered at each split, where usually $m \gg k$. This way, if few features dominate the rest in their contribution to the target variable, their contribution to the final model is decreased as now the chances for them to be selected for a split are reduced.

Another tree-based ensemble technique is called boosting when power of combining weak learners is leveraged.

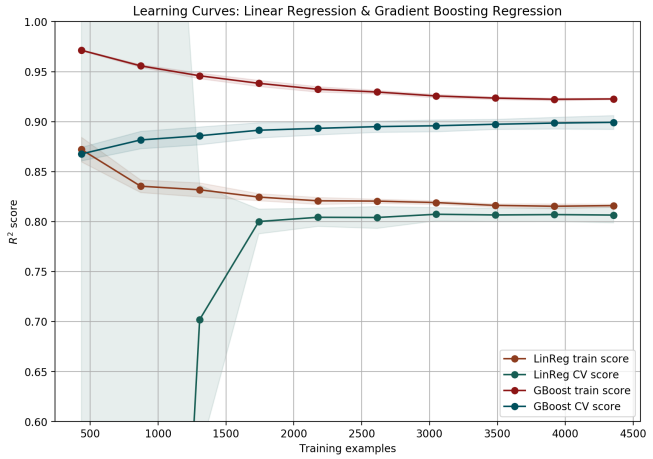

Fig. 2: R^2 as a function of training set size for LR and GB

Model	R^2 (train)	R^2 (CV)	MSE (train)	MSE (CV)	MSEcv/MSE_train	R^2 (test)	MSE (test)
Linear Regression	0.8154	0.8066	0.1839	0.1922	1.0452	0.8032	0.1998
LinReg + LASSO	0.8136	0.8033	0.1857	0.1955	1.0529	0.8023	0.2006
Random Forest	0.9554	0.8950	0.0444	0.1042	2.3442	0.9033	0.0981
Gradient Boosting	0.9200	0.8993	0.0797	0.1000	1.2545	0.9069	0.0945
Neural Network	0.8470	0.8278	0.1525	0.1750	1.1480	0.8325	0.1699

Table I: Accuracy and variance metrics for different methods

"Weak" in our case usually means shallow trees with only 3-4 levels of splitting nodes. This algorithm is called forward stagewise boosting and is also done in a greedy fashion [13]. To overcome this shortcoming, the modified version called Gradient Boosting (GB) is used when the current iteration tree is fit to the negative gradient of the loss function w.r.t. the previous tree. This tree is being used in the minimization procedure as done in stagewise boosting to learn the parameters of the final tree that is added to the previous iteration tree [13]. The most important hyper-parameters for GB is the number of trees, maximum depth of the tree, minimal leaf size, learning rate, and the loss function.

V. EXPERIMENTS & RESULTS

We start off with the linear regression (LR) using all features. The reported CV MSE is 0.19 (Table I) with larger test MSE (0.2). The variance given by MSE_{CV}/MSE_{train} is only 1.04 suggesting the model is generalizing well on new data. The problem with linear regression is that it underfits. The R^2 plotted as a function of training set size (Fig. 2) is flattened starting from about 3000 observations for both CV and training sets suggesting LR is reaching its predictability limit given our features and remain under desired performance level (around 0.95-0.98). When LASSO regularization is added to the LR, the MSE and R^2 remain almost unchanged which is an expected result considering we standardized all our real-valued features. The large observed bias is due to limitations of linear method since it is not accounting for nonlinear relationship between condo features and the selling price.

This made us try nonlinear learning methods, and the first one was neural networks (NN). There is no formula for building the "perfect" NN architecture. Many design decisions are empirical and based on past experiences using them. As a result, we experimented with various layer depths, neuron counts, and activation functions. We found that in general, deeper networks with smaller layers performed better than shallow networks with larger layers. This is supported by [9] which shows that deeper networks can represent more complex relationships between features, and by [10] which suggests that larger layers require more parameters which make them more likely to overfit the training data. Below is a highlight of 6 network architectures tested in this experiment (numbers inside the activation functions represent the number of nodes in a layer):

input →

1) ReLU(50) → ReLU(100) → ReLU(50)
2) ReLU(64) → Dropout(0.2) → ReLU(32) → Dropout(0.2) → ReLU(16)
3) ReLU(16) → Dropout(0.2) → ReLU(8) → ReLU(4)
4) ReLU(24) → Dropout(0.2) → ReLU(16) → Dropout(0.2) → ReLU(12) → ReLU(8) → ReLU(4)
5) LeakyReLU(16) → Dropout(0.2) → LeakyReLU(8) → Dropout(0.2) → ReLU(4)
6) LeakyReLU(64) → Dropout(0.2) → LeakyReLU(32) → Dropout(0.2) → LeakyReLU(16)

→ 1 output node

The NN initially suffered from significant overfitting (CV MSE was 10x of training MSE). This issue was addressed by adding dropout layers to the network and choosing the best configuration (5). As a result, the train-CV MSE difference dropped to below 15% (Table I), and the corresponding test R^2 and MSE became 0.83 and 0.17, respectively.

The next nonlinear method we employed was Random forest (RF). Initial default parameters produced a model with large unacceptable variance (around 30), and decreasing the maximal depth of the trees to 10 reduced overfitting and produced reasonable MSE for both CV and training sets. The RF implementation reduced MSE and raised prediction accuracy to a new level producing CV and test MSE as small as 0.1 (Table I). There still remained room to further generalize our model since MSE_{CV}/MSE_{train} was 2.34 suggesting our model still overfitted the data.

Gradient Boosting Regression (GB) finally provided most robust and accurate model leveraging non-linear nature of interaction between condo features and target variable reporting CV set MSE 0.1 and test MSE 0.09 with MSE_{CV}/MSE_{train} being 1.25 (Table I). To further improve the performance of GB model, we tried to find optimal hyper-parameters for the model by CV gradient search when

model is being fit sequentially to CV subsets with variable combinations of hyper-parameter (loss function, number of trees, maximal tree depth, minimal leaf size, and learning rate) values. The optimal set of hyper-parameters was least squared loss (ls), 200, 20, 10, 0.05. With these values, CV R^2 was 0.95 but variance raised up to 8.32 which was not acceptable. Further tuning of hyper-parameters and checking CV MSE, R^2, and MSE_{CV}/MSE_{train} gave us hyper-parameters $\{ls, 100, 3, 10, 0.05\}$ which was close to the default settings ($\{ls, 100, 3, 2, 0.1\}$) and gained only marginal improvement. The R^2 curve as a function of training set size (Fig. 2) suggests that our GB model reached good results in terms of both accuracy and generalization, especially when we compare it with LR curves. Another experiment we performed was to find a feature subset selected by LASSO (32 features out of 48) but it gained no improvement supposedly because GB algorithm already implicitly assigns importance to different features.

VI. DISCUSSION

In addition to testing the three models, we looked at the top features highly correlated with selling price of a condo. They were (in order):

1) number of bedrooms
2) number of bathrooms
3) total floor area
4) number of parking spaces
5) maintenance fees

We also noticed that the more expensive the condo was, the more subjective the price appeared to be (and the worse our models performed). Speaking about listing price that we intentionally excluded from the model, we note that it is highly correlated with our target variable, the selling price. The MSE between them is about 6 times smaller than our best MSE. One might think that the domain knowledge of real estate agents is very thorough in estimating home value but there is a paradox. When people want to sell or buy a home, they first look at the listing price and therefore the resulting selling price often is very close to the listing price, with listing price being a major driving factor. Our goal, on the other hand, was to come up with the emotion-free algorithm that uses only bare facts about property itself and external factors. Comparing metrics of our model to those achieved by others, we see that theres still room for improvement.

VII. CONCLUSIONS & FUTURE WORK

Our findings show that gradient boosting had the best performance, followed closely by random forest and neural networks. This makes sense because both algorithms are useful for non-linear modeling. The small data scale in our project lends itself more favorably to trees, whereas it makes it more likely for a neural network to overfit (resulting in slightly lower overall performance).

The models described in this paper can be further improved in future work through usage of additional training datasets and additional feature engineering. The federal interest rate has a direct effect on the supply of money and affordability of housing, which can affect the selling price. Over the timespan of the training dataset used in this study, the federal interest rate changed from a low of 0.5% up to a high of 1.75% which could significantly affect the selling price of a condo. An additional dataset that would improve the model is upcoming new condo developments. Large, growing cities often have new real estate being built. Additional inventory coming onto the market would affect existing condo prices negatively by increasing the available supply and alternatives for buyers.

Lastly, future work should focus on adding more temporal components to the model, for example through features such as listing date, "number of condos sold in last n days", and "n-day average sell price".

ACKNOWLEDGMENT

We'd like to thank Adina Dragasanu from RE/MAX Crest Realty for providing us the data that enabled our research.

REFERENCES

[1] R. John, "Simple Housing Price Prediction Using Neural Networks with TensorFlow", Medium.com, May 29, 2018

[2] H. Yu and J. Wu, "Real Estate Price Prediction with Regression and Classification" (2016)

[3] TensorFlow, https://www.tensorflow.org/api_docs/python/tf

[4] scikit-learn, https://scikit-learn.org/stable/

[5] matplotlib, https://matplotlib.org/

[6] S. Raghavan, "Create a model to predict house prices using Python", Jun 17, 2017

[7] D.Cock, "Ames, Iowa: Alternative to the Boston Housing Data as an End of Semester Regression Project", Journal of Statistics Education Volume 19, Number 3 (2011)

[8] S. Li, "Linear Regression in Python; Predict The Bay Area's Home Prices", TowardsDataScience.com, Oct 24, 2017

[9] R. K. Srivastava, K.Greff, J. Schmidhuber, "Training Very Deep Networks" , Nov, 2015

[10] "Explore overfitting and underfitting", https://www.tensorflow.org/tutorials/keras/overfit_and_underfit , Dec 2018

[11] A. Karpathy, "CS231n: Convolutional Neural Networks for Visual Recognition", http://cs231n.github.io/convolutional-networks/ , 2018

[12] https://developers.google.com/maps/documentation/geocoding/

[13] T. Hastie, R. Tibshirani, J. Friedman: The Elements of Statistical Learning. Data Mining, Inference, and Prediction. 2013, Springer, 2nd edition, 745p.

[14] M. eh, M. Kilibarda, A. Lisec and B. Bajat: Estimating the Performance of Random Forest versus Multiple Regression for Predicting Prices of the Apartments. ISPRS Int. J. Geo-Inf. 2018, 7(5), 168

[15] V. Moosavi: Urban Data Streams and Machine Learning: A Case of Swiss Real Estate Market. 2018. Online publication, https://arxiv.org/pdf/1704.04979.pdf