Melbourne Airbnb Price Prediction

Tiancheng Cai[*] Stanford University

Kevin Han[†] Stanford University

Han Wu[‡] Stanford University

Abstract

Our project tries to build a price prediction model for Airbnb listings in Melbourne. We leverage several regression methods and compare their performance. [4]

1 Introduction

Airbnb is the leading platform in providing lodging and tourism experience. Inside Airbnb's business logic, pricing is probably the most important for hosts and customers. Ensuring fair pricing directly affects booking activities, and also matters to the well-being of the e-commerce environment. Thus, studying the reasonable forecast and fair suggestion of prices of Airbnb listings can have huge real-life values and may generalize to other applications as well.

Melbourne, Australia was ranked the best city to live in the world by The Economist Intelligence Unit for seven consecutive years until 2018. With plenty of listing and booking activities, Melbourne serves as a great example for the study of Airbnb pricing.

In this project, we build a price prediction model of Airbnb listings and make comparisons between different methods. The input to our algorithm is Melbourne Airbnb Open Data [Mel, 2018] which contains continuous / categorical data as well as rich text data (descriptions, reviews) about listings. We then use traditional ML methods (linear regression, ridge regression, support vector regression, random forest regression, gradient boosting) and four neural networks to output the predicted prices of listings.

2 Relevant Works

Prior works on rental price prediction based on Airbnb data are deficient in terms of evaluation metrics and performance. Tang and Sangani [2015] work on the task of price prediction for San Francisco Airbnb listings. They turn the regression problem into a binary classification problem by spiliting the price according to the median, which effectively reduces the difficulty of the task. In a more recent work, Kalehbasti et al. [2019] works on price regression for Airbnb listings in New York. They use a range of methods including tree-based models, SVR, KMC, NN, etc and integrate sentiment analysis into their model. While they claim to achieve an highest R^2 of 0.7246, they evaluate their metrics (R^2 and MSE) on the logarithmic scale of price instead of the original scale.

Our project works on the original price regression problem, without transforming to a classification problem or evaluating on logarithmic scale. Besides traditional machine learning methods, we would integrate text data (from descriptions and reviews) into our model. To our best knowledge, there is no existing literature that uses text data as input variables for Airbnb price prediction.

[*]SUNet ID: caitch

[†]SUNet ID: kevinwh

[‡]SUNet ID: hanwu71

[4]Code available at https://drive.google.com/open?id=1D32jVpSfvEYCDCoVt6FYxS98KVFhp3Fm. Please use Stanford account to access.

3 Dataset

We use the Melbourne Airbnb Open Data for December 2018. In particular, for traditional machine learning methods, we only use non-text data in the file cleansed_listings_dec18.csv. For training neural network, we additionally employ text data that are stored in cleansed_listings_dec18.csv and reviews_dec18.csv.

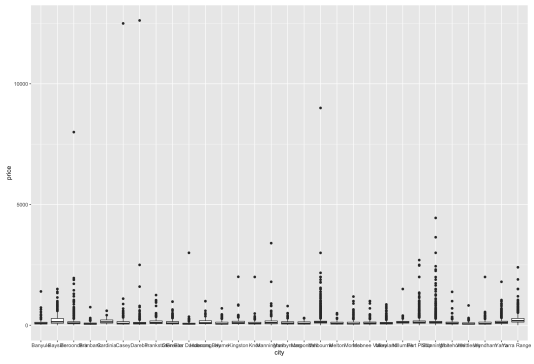

Figure 1: Barplot of city against price

Initially, the dataset contains 84 columns, which consist of 83 features and 1 response column. We perform the following data pre-processing steps:

1. We drop features that are apparently not related to price, for instance, host_id and scrape_id. We also drop features that contain >= 20% missing values. We end up with 44 non-text features.
2. Figure 1 shows a barplot of city against price. We see that there are some apparent outliers in the dataset. We remove all the listings with price higher than $1000 or with price of $0.
3. We perform specific processing on these columns: host_verifications, amenities and host_since. We convert host_vefifications feature from a list containing verification labels to the length of the list. We parse amenities from string to categorical variables. We calculate the number of days between hist_since and the date the data was extracted.
4. We remove rows with missing values and construct one-hot encoding for categorical features.
5. We obtain the text data by merging the descriptions and reviews of listings from the two CSV files.

For non-text (continuous/categorical) data, We end up with 22725 entries and 155 features excluding price. We split the dataset into 18179 (70%), 2273 (15%) and 2273 (15%) entries for training, dev and test sets respectively.

For text data (descriptions and reviews), we end up with 485416 entries, consisting of 388175, 51444 and 45797 entries for training, dev and test sets respectively. (We split the dataset with the same assignment as for non-text data, and every listing may have a number of reviews).

To prevent overfitting, we perform feature selection by LASSO [Tibshirani, 1996] for non-text data. Lasso (least absolute shrinkage and selection operator) solves the following problem

$$\arg\min_{\beta} \frac{1}{N} \sum_{i=1}^{N} (y_i - x_i^T \beta)^2 + \lambda ||\beta||_1$$

Lasso leads to sparse β hence can be used to select features. To accommodate potential clusters in data, we run LASSO on three parts of the data: listings with price under $250, between $250 and $500 and above $500. We run Lasso with 5-fold cross validation and select features on each part. Finally, we take the union of three sets of selected features. We end up with 80 out of 155 features. They all make intuitive sense as being potentially useful for predicting price.

4 Methods

We first consider traditional ML methods using continuous and categorical features. We use linear regression as baseline, and employ LASSO, ridge regression, support vector regression, random forest regression and gradient boosting. We fit two models for the above methods–one using all features and another using only selected features.

We then employ deep learning and train four neural networks as detailed in the following section.

4.1 Linear and Ridge Regression

As a baseline model, we run linear regression using encoding described above. Ridge regression adds an ℓ_2-penalty to the usual least square objective function with shrinkage parameter λ. Specifically, ridge regression solves the following problem

$$\arg\min_{\beta} \sum_{i=1}^{N} (y_i - x_i^T \beta)^2 + \lambda ||\beta||_2^2$$

4.2 Random Forest and Gradient Boosting Regression

Random Forest is an ensemble method that aggregates many decision trees by performing bootstrap aggregation to help decorrelate trees [Hastie et al., 2009]. Gradient boosting is another ensemble method that combines decision trees. While random forest builds trees in parallel, gradient boosting builds trees in a serial manner, where each tree tries to correct the mistakes of the previous one. Specifically, the Gradient Boosting algorithm is the following [Hastie et al., 2009]:

Algorithm 1 Gradient Tree Boosting Algorithm

1: Initialize $f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$

2: For m = 1 to M:

(a) For $i = 1, 2, \cdots, N$ compute

$$r_{im} = - \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f=f_{m-1}}$$

(b) Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, 2, \cdots, J_m$.

(c) For $j = 1, 2, \cdots, J_m$ compute

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma).$$

(d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{I}(x \in R_{jm})$.

3: Output $\hat{f}(x) = f_M(x)$.

4.3 Support Vector Regression

Specifically support vector regression solves the following optimization problem

$$\begin{aligned} \arg\min_{w,b,\xi_i} \quad & \frac{1}{2}||w||^2 + C \sum_{i=1}^{N} \xi_i \\ \text{subject to} \quad & y_i - w^T \phi(x_i) - b \leq \epsilon + \xi_i \quad \forall i = 1, N \\ & w^T \phi(x_i) + b - y_i \leq \epsilon + \xi_i \quad \forall i = 1, N \\ & \xi_i \geq 0 \quad \forall i = 1, N \end{aligned}$$

where C controls the amount of regularization and ϵ is the margin. We can also kernalize the method.

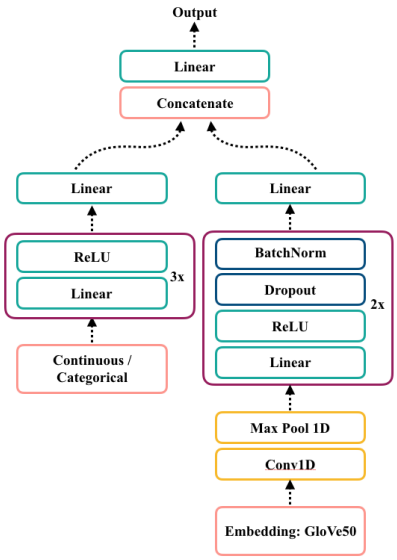

Figure 2: Combined Neural Net Structure

Figure 3: Predictions VS Actual Prices for Gradient Boosting

4.4 Neural Network

4.4.1 Using continuous and categorical features

We train a four-layer neural network, using three fully-connected hidden layers of 155, 15 and 15 hidden units with ReLU activation. The left branch of Figure 2 shows the structure.

4.4.2 Using only description data

We train a five-layer neural network that integrates description data by adding a word embedding layer. We use GloVe 50 [Pennington et al., 2014] word vectors as initialization of embedding layer. We then use one convolution layer with window size 5, output dimension 128 and max pooling followed by two dense layers with output dimension 128, 32, dropout rate 0.3, 0.2, batch normalization and ReLU activation. The right branch of Figure 2 shows the structure.

4.4.3 Using all text data(descriptions and reviews)

The model structure is the same as the previous section. The only difference is the inputs. For each review of the listing, we concatenate the review with its corresponding description. Since one listing usually has many reviews, we enlarge the number of training samples. When we split the data into training, validation and test set, we split based on listing ID so that we make sure that there is no overlap of listings among these three sets. Hence, for this task, the training, validation and test set are not the ones we used for other models. However when we calculate MSE, we do the following. For each listing in the set, there are several description-review pairs. Our model predicts a price for each pair. We use the average of such predictions as the predicted price for this listing and calculate MSE accordingly. As a result, it is reasonable to compare the performance of this model with others.

4.4.4 Using all features (continuous/categorical/description data)

The combined pipeline structure is shown in Figure 2. The left part uses continuous and categorical features, and the right part uses descriptions. Note that here we do not include reviews since each listing corresponds to many reviews and if we concatenate all of them, the input dimension is so large that we need many more parameters in our model, which results in quick overfitting. /;

Machine Learning Models		Test MSE	Test R2	Training MSE
No Feature Selection	Linear Regression	6460.7976	0.5046	7087.0750
	Ridge Regression	6460.7368	0.5046	7087.4613
	Random Forest	4513.5144	0.6539	761.5576
	Gradient Boosting	4024.7052	0.6914	2468.5471
	Support Vector Regression	5246.2777	0.5977	4878.6639
Feature Selection with LASSO	Linear Regression	6800.5157	0.4786	7484.7884
	Ridge Regression	6800.8531	0.4786	7485.6137
	Random Forest	4422.7124	0.6609	789.7775
	Gradient Boosting	4156.5544	0.6813	2392.1191
	Support Vector Regression	5459.5413	0.5814	5296.0410
Deep Learning Models		Test MSE	Test R2	Training MSE
Original Features	Four-Layer Feedfowrad NN	4632.5926	0.6448	4197.0896
Text Data Only	Using Description	8523.1239	0.3465	1599.9195
	Using Description + Review	5467.3576	0.4717	2301.7509
Original Features + Text	Combined model	4526.6191	0.6529	1023.2743

Table 1: Results from traditional ML and DL Models

5 Experiments and Results

We use mean squared error (MSE) and R^2 score as evaluation metrics. Results from our models are shown in Table 1. Figure 3 shows the actual prices (x-axis) against our best predictions (y-axis) by Gradient Boosting on test set. The red line represents the perfect prediction (x = y).

For the traditional machine learning methods, we use Scikit-learn library [Buitinck et al., 2013]. We use grid search for hyper-parameter tuning, with 3-fold cross validation on the train/dev set. For ridge regression, we select the shrinkage parameter $\lambda = 100$. For Random Forest Regressor, we select max feature=10, unlimited depth and use 1000 estimators. For Gradient Boosting Regressor, we select max depth=7, max features=6 and use 200 estimators. For Support Vector Regressor, we select C = 100.0, $\epsilon = 0.5$, and use RBF kernel. We use Keras [Chollet et al., 2015] when implementing the deep learning methods and referred the code at [Robinson, 2018], [Furuta, 2018] and [Chollet et al., 2018]. For the neural network with continuous/categorical features, we use batch size = 1000 and for other networks we use batch size = 32. We set learning rate = 0.001 and use Adam Optimizer [Kingma and Ba, 2014]. We use early stopping against dev set to mitigate overfitting.

We find that gradient boosting with all features perform the best among all models. Random forest has the second best performance, and we see that feature selection improves the performance of Random Forest in particular.

As for deep learning models, the neural network using all features (continuous, categorical and descriptions) achieves comparable accuracy. Using additional text input has contribution to performance in the presence of continuous and categorical features. Neural network using text features alone also shows reasonable performance, which is beyond our expectation.

6 Discussion and Future Work

From Figure 3 we see that our best model tends to underestimate the price of listings with higher prices. If we instead consider listings with price <= $500 only, we would achieve significant improvements. (e.g. MSE on test set with Deep Learning model using only descriptions and reviews drops to 2894.2764). However, we choose to evaluate on listings with price <= $1000 for more generalizability. We see from the table that random forest is more likely to suffer from overfitting. Also, incorporating reviews does help when we use text data only as our inputs.

As for future work, we will consider more careful feature selection and trying out two-step modeling, which would divide training sets into K groups based on price range and build separate models for each group. For prediction it would classify group label first and then run price regression. As for deep learning models, we would like to try out different word vectors (such as word2vec) and more complex NLP models. Lastly, we would like to see the transferability of our model when applied to different cities.

7 Contributions

We equally contributed to the project. Specifically, we all wrote code and ran experiments for traditional machine learning methods and deep learning methods. We made the poster and wrote the final report together.

References

Melbourne airbnb open data. https://www.kaggle.com/tylerx/melbourne-airbnb-open-data, 2018.

L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

F. Chollet et al. Keras. https://keras.io, 2015.

F. Chollet et al., 2018. URL https://keras.io/examples/pretrained_word_embeddings/.

Furuta, 2018. URL https://gist.github.com/furuta/eb2453a0ad13ca51b780caef5f3c95c7.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer-Verlag, New York, 2009.

P. R. Kalehbasti, L. Nikolenko, and H. Rezaei. Airbnb price prediction using machine learning and sentiment analysis, 2019.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014.

J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

S. Robinson. Predicting the price of wine with the keras functional api and tensorflow, 2018. URL https://medium.com/tensorflow/predicting-the-price-of-wine-with-the-keras-functional-api-and-tensorflow-a95d1c2c1b03.

E. Tang and K. Sangani. Neighborhood and price prediction for san francisco airbnb listings. *CS 229 Final Project Report*, 2015.

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996. ISSN 00359246. URL http://www.jstor.org/stable/2346178.