Final Project Report

Group 7: Yunze Pan (yp2599), Pavan Nekkanti (ln2460), Nitya Krishna Kumar (nk2963), John Nguyen (jn2814), Tengteng Tao (tt2830)

Introduction

Estimating the price of the property is never an easy task. Due to the complexity of factors that influence the house's price, it is difficult to train employees in a real estate investment firm to become experienced appraisers with limited amounts of time and effort, not to mention the fact that appraisers with years of experience can potentially make biased judgements. Predictions between different individual appraisers greatly vary. In order to facilitate the process of estimating the price of the properties, and predict the price accurately by minimizing the errors brought by humans, machine learning models can be deployed to estimate the value of houses. Our goal is to assist appraisers to better understand the relation and patterns among the collected data so that they can make predictions on the price of properties more consistently and accurately.

Data Description

For this project, we use data from *https://github.com/jsrpy/NYC-Property-Regression/* There are 1883 observations of house transactions. '*tx_price'* is the target variable. Table 1 in the Appendix is a description of the various independent variables.

Exploratory Data Analysis and Data Cleaning

Through our analysis of both numerical as well as categorical variables, below are insights we have gathered on the data:

- Correlation matrix indicates that there is a relatively strong relationship between features related to physical characteristics of a house (i.e. sqft) and the house price. Surprisingly, the environmental attributes of a house seem to have a very weak relationship with the price. Compared with the environmental attributes, residents' demographics in a neighborhood seem to be more correlated with the price.
- *Insurance* and *property_tax* are features that seem to portray a linear relationship with our target variable.
- Our analysis of categorical data shows that single-family homes are more expensive. It's very likely that single-family homes may have larger floor area.

Missing Data

We can clearly find that 'exterior_walls', 'roof' and 'basement' have missing values. For the feature 'basement', we can find that the value is either 1 or Nan. Thus, the missing value here stands for no basement for the building, and we can simply replace the missing value with 0 here.
It is hard to find any relationship between missing values in feature 'exterior_walls' and missing values in feature 'roof'. Thus, we need to discuss them separately. After calculating the frequency of each value in feature 'roof', we found that value 'Composition Shingle' appeared 1179 times out of 1883 data points. Accordingly, using mode to replace missing values is a good choice here. For the feature 'exterior_walls', we found that this feature has a correlation with another feature 'property_type'. Exterior walls' type: 'Siding(Alum/Vinyl)' appeared 345 times out of total 503 times when the property type is single family.

In addition, when the property type is an apartment, exterior walls' type: 'Brick' value appeared 381 times out of 687 times. Thus, we can use the feature 'property_type' to predict the missing values in feature 'exterior_walls' by setting that when the 'property_type' is apartment, then we predict the missing values in 'exterior_walls' as 'Brick'; when the 'property_type' is single family, then we predict the 'Siding(Alum/Vinyl)' in feature 'exterior_walls'.

Data Handling

Feature Expansion:

- 'house_age' = 'tx_year' - 'year_built'

Missing Data:

- *basement*: number of basements → replace missing value with 0.
- *roof*: roof material → replace missing value with mode.

Categorical Data:

- Ordinal Encoding for Apartment/Condo/Family Home(0) and Single Family(1).
- similar variable names will be combined.

Multicollinearity:

- 'nightlife', 'cafes', 'shopping', 'spas', 'active life': highly correlated variables will be removed.

Outliers:

- 'insurance' > 1,200, 'property_tax' > 4,000: they are detected as outliers based on the scatter plot and the box plot and will be removed.

Log Transformation:

- 'tx_price': reduce the skewness.

Standardization:

- use the Standard Scaler function to avoid the bias.

Train-Test Split

- The data was split into training and testing in a 80:20 split.

Evaluation Metrics

We then follow five main steps to predict the house prices. 1) model selection: select ML algorithms. 2) random search: set model hyperparameters. 3) cross validation: set a 5-fold cross validation. 4) model training: tune hyperparameters with grid search & cross validation. 5) evaluation: explore the performance achieved by different machine learning models using three different evaluation metrics. Evaluation for a regression problem: R square score, MSE, and MAE. Three formulas are shown below.

$$R^2(y,\hat{y}) = 1 - \frac{\sum_{i=0}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{n}(y_i - \bar{y})^2}. \quad MSE(y,\hat{y}) = \frac{1}{n}\sum_{i=0}^{n}|y_i - \hat{y}_i|^2. \quad MAE(y,\hat{y}) = \frac{1}{n}\sum_{i=0}^{n}|y_i - \hat{y}_i|.$$

Analysis of Machine Learning Algorithms

For a conclusive list of the results of all models, please refer to Table 2 in the Appendix.

<u>Baseline Model</u>

First, we create a baseline regression model using the mean of the house price so that we will compare other various machine learning algorithms' performance with this simple baseline model.

Ridge Regression, Lasso Regression, Elastic Net

First, we implemented three different linear regression models to see if we can fit our data into a linear model. We trained Ridge, Lasso and Elastic Net regression models and tuned by using grid search technique and 5 fold cross validation. The performance of all three models is significantly lower compared to all the other models which indicates that we cannot use linear regression models to fit our data.

Random Forest

We used the Random Forest Regressor model to solve this problem and decided to tune the model using the three parameters: number of trees (n_estimators), max number of features (max_features) and depth of individual trees (max_depth). We used grid search technique to find the appropriate values for these parameters and used 5 fold cross validation to select the best model. We found that the property_tax, tx_year and sqft are three most important features which seems to be inline with our initial data analysis and also makes more sense in a real world scenario. The r2 score on the test set is 0.76 which is much higher than the linear models.

Gradient Boosting

We used the Gradient Boosting Regressor model to implement one of the boosting algorithms. We used three parameters: shrinkage factor (learning_rate), number of trees (n_estimators) and depth of individual trees (max_depth) to tune the model and used 5 fold cross validation to select the best model. We found that the property_tax, tx_year and sqft are three most important features which are the same as the important features in the Random Forest model. The r2 score on the test set is 0.77 which is almost comparable to the random forest regressor.

XGBoost

Another important model we applied here is the XGBoost regressor. XGBoost stands for Extreme Gradient Boosting. XGBoost is an optimized distributed gradient boosting library designed to be highly efficient and flexible. It is based on the Gradient Boosting Framework and provides a parallel tree boosting to make predictions efficiently. For this program, since we have applied the Random Forest and Gradient Boosting Models, by comparing those linear models we applied at the beginning with the random forest model, we can find a massive increase in both the model score, r2 score, MSE, and MAE. Thus, the XGBoost regressor, an improvement of the traditional gradient boosting tree model, is highly possible to be a better model with an even higher score.

Accordingly, we implemented the XGBoosting Regression with this assumption and trained the model with train data we had split before. We implemented Grid Search to tune hyperparameters and utilized cross-validation with five folders to avoid overfitting. We tuned the n_estimators, learning_rate, and max_depth for the hyperparameters. We gave two values of each hyperparameter and tried every combination to find the best fit on train data. Besides, with the help of 5 folders cross-validation, we can avoid overfitting efficiently, which promises a reliable prediction of our test data.

Based on the training result, we can see that the best combination of hyperparameters here is: learning_rate: 0.1, max_depth: 7, n_estimators:150. With this estimator, we can achieve accuracy on training data up to 99.49% and on test data up to 76.84%. We also calculated the corresponding R2 score, MSE, and MAE. For the training data, the R2 score is 0.99, with MSE equaling 0.005 and MAE equaling 0.05. For the test data, the R2 score is 0.76, with MSE equaling 0.233 and MAE equaling 0.334.

Although our evaluation metric on test data is still lower than the training data, we can still see a vast improvement compared to the gradient boosting and random forest model we applied before.

LightGBM Regressor

Since we had applied XGBoost before and got a relatively good result, we decided to try the LightGBM regressor, similar to the XGBoost we applied before. However, a majority difference is that XGBoost trees grow depth-wise, but trees grow leaf-wise in LightGBM.

Similarly, we also implemented hyperparameter tuning and cross-validation here. We will also tune hyperparameters: n_estimators, learning_rate, and max_depth in the LightGBM regressor. To avoid overfitting, we still use cross-validation with five folders and train data to train the model.

Based on the training result, the best combination of hyperparameters in LightGBM Regressor is: learning rate equals 0.1, max depth equals 7, and n estimators equal 100. For this best estimator, the accuracy for the training data is up to 95.2%, and for the test data is 78.1%.

By comparing with the XGBoost regressor, we can see that although the accuracy of training data is lower, we achieve higher accuracy on test data, which means we went one step further in fighting overfitting. The Evaluation metrics also illustrated similar results. For the training data, the R2 score is 0.95, with MSE equaling 0.048 and MAE equaling 0.165. For the test data, the R2 score is 0.78, with MSE equaling 0.221 and MAE equaling 0.334.

Overall, based on the gradient boosting method, both the XGBoosting regressor and the LightGBM regressor performed well in our data set. The LightGBM is better since it achieved a higher accuracy on test data.

CatBoost Regressor

Since we had some categorical variables, we decided to try applying the CatBoost algorithm to the data. The data was first scaled using standard scaler then a randomized Search was used with 5-fold cross validation was run to tune the following hyperparameters: learning_rate: [0.03, 0.1], max_depth: [4, 6, 10], and l2_leaf_reg: [1, 3, 5]. After training, the best parameters were {'max_depth': 7, 'learning_rate': 0.03, 'l2_leaf_reg': 5}. The model yields an R squared similar to XGBoost and LightGBM on the test dataset - approximately 0.7959. The model also has low MSE and MAE, though not as low as Random Forest or Gradient Boosting. From the feature importance plot, it can be seen that *tx_year* and *sqft* played a large role in predicting the price of the house. It can also be found that *roof* type also plays an important role in predicting the price of the house.

Support Vector Regression

Compared with SVM, Support Vector Regression (SVR) tries to solve a regression problem. It is especially helpful for us to predict house prices if the trends are not linear. In SVR, a hyperplane line is defined to predict the continuous value so that our data will be transformed to a lower dimension. There are different parameters in the SVR model. We choose two parameters to tune which are the penalty term C and the gamma value. We use the radial basis function kernel (rbf) to train the dataset. Larger values of C will lead to the larger variance and lower bias and larger values of gamma will lead to higher bias. According to the results, setting gamma as ‘auto’ will give us the best score. However, SVR doesn’t perform much better than other machine learning algorithms.

Deep Neural Network

Keras tuner library was imported for hyperparameter tuning with Random Search. A regression model with 3 hidden layers and 1 output layer which predicts sales price with no activation function was used with objective loss function as mean squared error. The hyperparameters tuned are the number of neurons per hidden layer ([48, 528] in steps of 24), activation function ['relu', 'linear']. The data did not overfit, therefore regularization techniques such as dropout or batch normalization were not implemented as they appeared to lower accuracy metrics. The results of the neural network are poor compared to the classical machine learning methods. Although the training data performed higher than the test data, the accuracy and loss training plots did not show overfitting in between the test and validation data shown in figure 5. Other optimizers such as RMSprop and SGD were explored but Adam was chosen and appropriate for regression problems. The best model parameters were 312 neurons, 72 neurons, 504 neurons in the first 3 hidden layers and then linear activation functions respectively which created a linear model. To introduce a non-linearity baseline model, a simple model with 48 neurons for 3 layers with Relu activation was also explored and reported worse results. LeakyRelu was also explored to combat a potential vanishing gradient but the performance was degraded so the optimal hyper parameters were confidently tuned. Overall, the DNN performed worse but could potentially improve with more training time and more hyperparameters.

Conclusion and Future Work

Based on the results of the various models we tried, it seems that Random Forest, Gradient Boosting, XGBoost, LightGBM, and CatBoost retrieve the similar R squared values. However, the Random Forest and Gradient Boosting return much lower MSE and MAE values than the rest. Since the low MSE, MAE values are supported by relatively high R squared values, we can assume that this may not be a case of underfitting. There does not seem to be much of a difference between XGBoost, LightGBM, and CatBoost regressors. This may be due to the fact that the dataset was relatively simple and clean. As there were not many categorical variables, CatBoost did not perform much better. Due to the size of the data, there was also not much difference between XGBoost and LightGBM. Furthemore, the implementation of a Deep neural network does not appear to be a good candidate for regression with a relatively small dataset as it had a much lower R square value and higher error.

As mentioned before, all three of the initial regression models retrieved very low R squared values. All perform similarly and do not seem to be complex enough to fully learn the data. Support Vector regression retrieves better results than these initial regression models however, the R squared value is still less than the boosting methods. This method also has low train R squared values.

In the future, we can find more house sale datasets for the real estate market in various cities. In addition, we may analyze whether the seasonality effects will contribute to predicting the house price. Other future work includes using feature importance to choose important variables, using feature expansion to transform and combine features, more training time and compute clusters for Neural Network, larger Datasets and datasets from various locations, AutoML for comparison.

Appendix

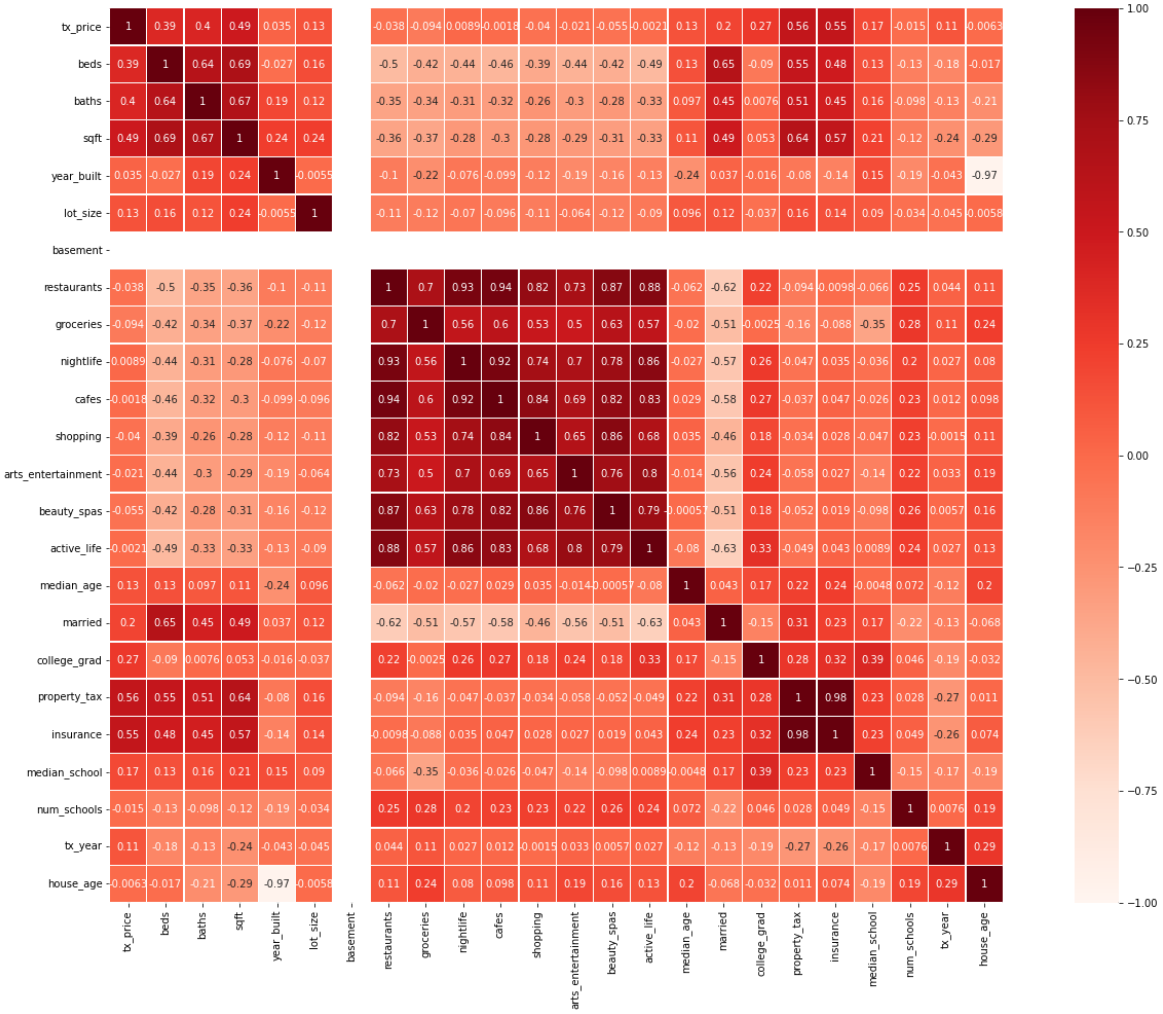

Figure 1: Correlation matrix

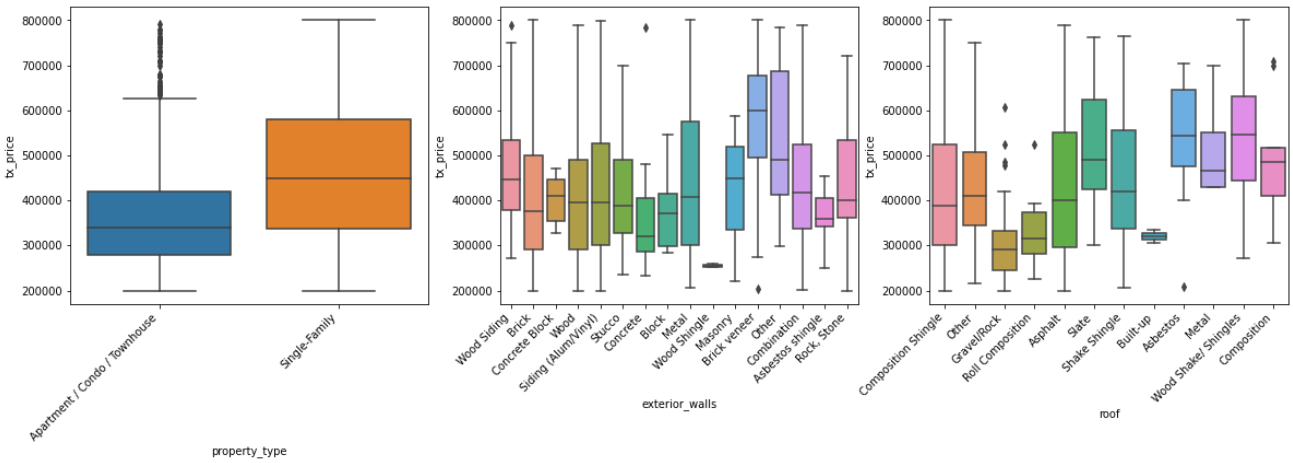

Figure 2: Categorical Data

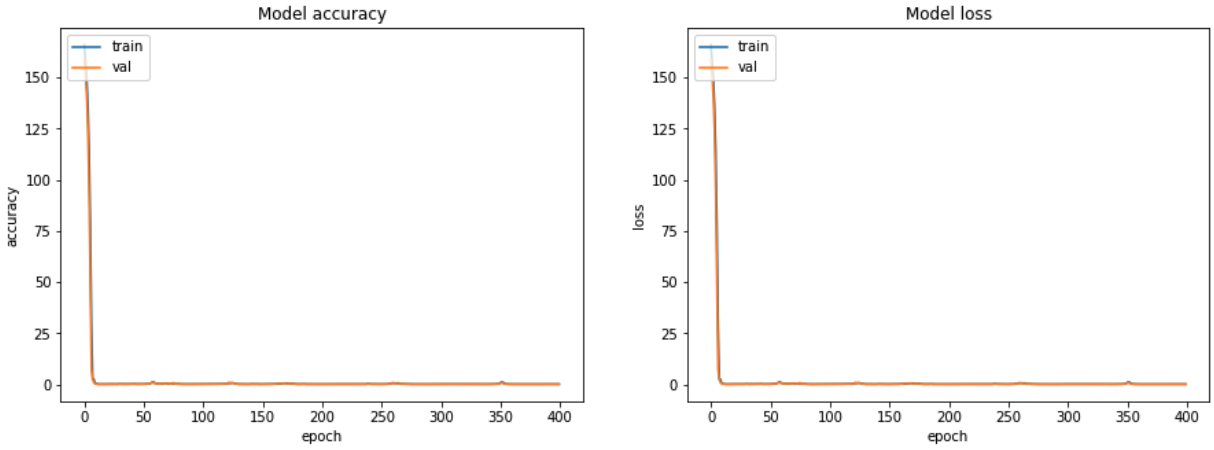

Figure 3: DNN Training Metrics

Property attributes		
Name of the feature	**Description**	**Data type**
beds	number of bedrooms	numeric
baths	number of bathrooms	numeric
sqft	floor area in square feet	numeric
lot_size	outside area in square feet	numeric
basement	basement? (yes/no)	binary
exterior_walls	material for walls	categorical
roof	material for roof	categorical
property_type	Apartment/Single-Family?	categorical

Demographic and environmental attributes		
Name of the feature	**Description**	**Data type**
median_age	median age of a neighborhood	numeric
married	% of married	numeric
college_grad	% of graduated college	numeric
restaurants	number of restaurants nearby	numeric
groceries	number of grocery stores nearby	numeric
nightlife	number of nightlife places nearby	numeric
shopping	number of shopping stores nearby	numeric
arts_entertainment	number of art & entertainment places nearby	numeric
beauty_spas	number of beauty and spa spots nearby	numeric
active_life	number of gyms nearby	numeric
median_school	median score of public schools nearby	numeric
num_schools	number of public schools nearby	numeric

Other important attributes		
Name of the feature	**Description**	**Data type**
year_built	year that house was built	numeric
tx_year	year that transaction was made	numeric
property_tax	monthly property tax	numeric
insurance	monthly insurance	numeric

Table 1: Data Insight

Model	**Train R^2**	**Train MSE**	**Train MAE**	**Test R^2**	**Test MSE**	**Test MAE**
Baseline	0.0	0.1257	0.2978	-0.0062	0.1277	0.2958
Ridge	0.5185	0.0605	0.2000	0.4300	0.0724	0.2125
Lasso	0.5141	0.0610	0.2011	0.4359	0.0716	0.2110
Elastic Net	0.5122	0.0613	0.2017	0.4389	0.0712	0.2105
Random Forest	0.9361	0.0080	0.0692	0.7561	0.0309	0.1309
Gradient Boosting	0.8819	0.0148	0.0930	0.7691	0.0293	0.1249
XGBoost	0.9949	0.0051	0.0508	0.7684	0.2339	0.3447
LightGBM	0.9519	0.0480	0.1648	0.7810	0.2211	0.3339
CatBoost	0.9621	0.0379	0.1494	0.7933	0.2087	0.3308
SVR	0.7852	0.0269	0.1271	0.6549	0.0438	0.1619
DNN	0.4411	0.0703	0.2125	0.3538	0.0820	0.2251

Table 2: Train and test results from all models