Project goal: Predicting condominium price using data from webscraping

1 Describe three machine learning algorithms used in the project

1.1 Ridge Regression

In order to understand ridge regression, we firstly need to understand the linear regression. Linear regression is one of the simplest ways to predict output $\hat{y}$ using a linear function of input features.

$$\hat{y} = w[0] \times x[0] + w[1] \times x[1] + \ldots + w[n] \times x[n] + b \qquad (1.1)$$ [1]

The equation 1.1 shows the linear model based on the n number of features. *w[0]* is a slope of *x[0]* and *b* represents the intercept. Linear regression algorithm tries to optimize *w* and *b* such that it minimizes the cost function (1.2). The cost function can be written as sum square of the errors.
Linear regression - objective: minimize cost function below:

$$\sum_{i=1}^{M} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{M} \left(y_i - \sum_{j=0}^{p} w_j \times x_{ij} \right)^2 \qquad (1.2)$$

Ridge regression tries to reduce model complexity and prevents over-fitting by adding a penalty equivalent to square of the magnitude of the coefficients to the cost function (1.3).

Ridge regression - objective: minimize cost function below:

original linear regression

$$\sum_{i=1}^{M} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{M} \left(y_i - \sum_{j=0}^{p} w_j \times x_{ij} \right)^2 + \lambda \sum_{j=0}^{p} w_j^2 \qquad (1.3)$$

penalty term added

The penalty term λ regularizes the coefficients so that if the coefficients w_j take large values, the cost function is penalized. When λ is 0, the cost function (1.3) becomes similar to the linear (1.2).

By increasing λ, the overall cost will increase. Therefore, to minimize the cost, the weights w_j will need to be reduced, which then helps to reduce model complexity and multi-collinearity by giving lesser weights to low important features.

[1] https://towardsdatascience.com/ridge-and-lasso-regression-a-complete-guide-with-python-scikit-learn-e20e34bcbf0b

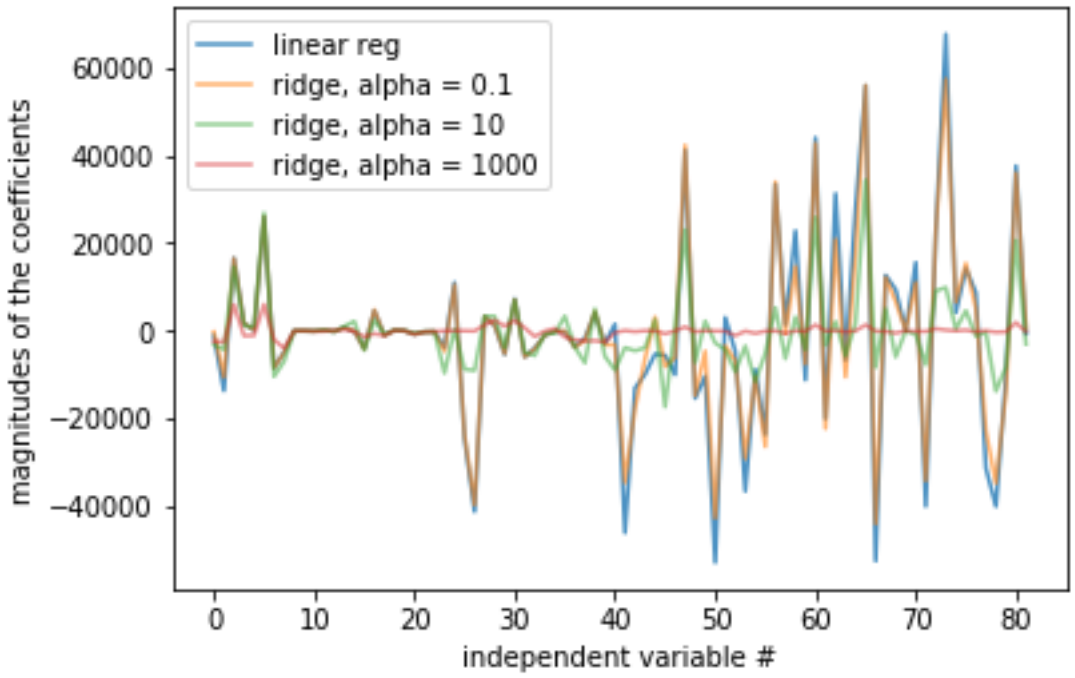

Figure 1.1.1: weights w_j with different shrinking coefficients (real data from this project)

In Python sklearn.linear_model.Ridge[2], the λ parameter is called 'alpha'. Figure 1.1.1 shows an example of shrinking coefficient magnitude using in Ridge regression algorithm. At alpha 0.1 (orange line), when the coefficients are less restricted, the magnitudes of the coefficients are almost same as of linear regression (blue line). For higher value of alphas, the magnitudes of coefficients w_j are considerably less compared to linear regression case.

1.2 Random Forest Regression

Random forest is an ensemble learner built on decision trees. Decision trees are intuitive ways to classify or label objects based on efficient binary splitting. In a tree, each question will cut the number of options approximately by half, each node in the tree splits the data into two groups using a cutoff value within one of the features. For example, if we want to build a decision tree to classify an animal, we might construct the one shown in figure 1.2.1.

[2] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html

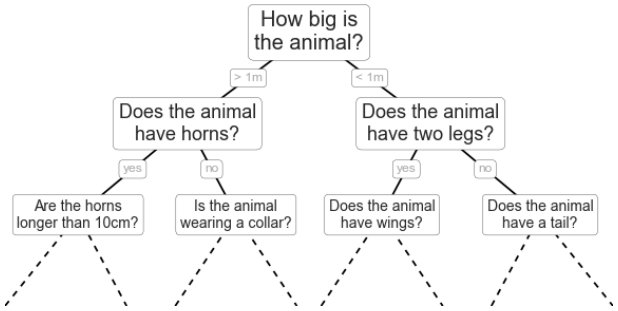

Figure 1.2.1: Decision tree model[3]

A simple decision tree will iteratively split the data along one or the other axis according to some quantitative criterion, and at each level assign the label of the new region according to a majority vote of points within it. Each decision is based on attribute selection measure; Information Gain, Gain Ratio, and Gini Index[4]. At each level, every region is again split along one of the two features, except for nodes that contain all of one color (figure 1.2.2).

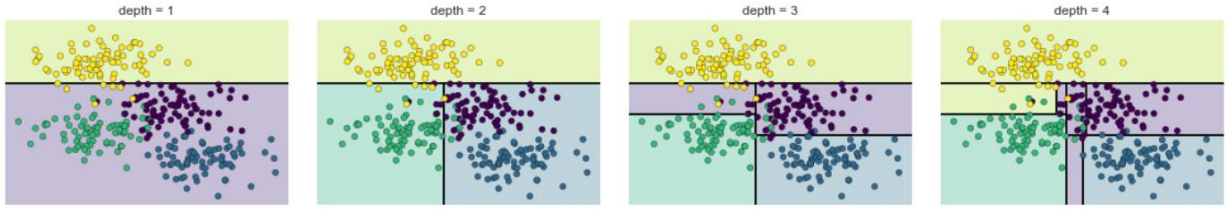

Figure 1.2.2: the first four levels of a decision tree classifier[3]

Overfitting is generally the main problem of decision tree (low bias, high variance). When there are strong predictors in the data set, the trees will always select that strong predictor at very first stage of splitting and result in overfitting.

Random forest tries to fix this issue by randomly select *m* predictors from total *p* predictors in the dataset to avoid split using only strong predictor in the data. This increases diversity in the forest leading to more robust overall predictions. When it comes time to make a prediction, the random forest takes an average of all the individual decision tree estimates. For example, the problem in this paper, we are predicting a continuous value of condominium price.

[3] https://jakevdp.github.io/PythonDataScienceHandbook/05.08-random-forests.html
[4] http://www.ijoart.org/docs/Construction-of-Decision-Tree--Attribute-Selection-Measures.pdf

1.3 Gradient Boosting Regression

The intuition behind gradient boosting algorithm is to repetitively fit the patterns in residuals and strengthen a model with weak predictions and make it better. Algorithmically, we are iteratively minimizing our loss function, such that loss reaches its minimum. The iteration stops when we reach a point that residuals do not have any pattern that could be modeled. The process can be visualized in figure 1.3.1.

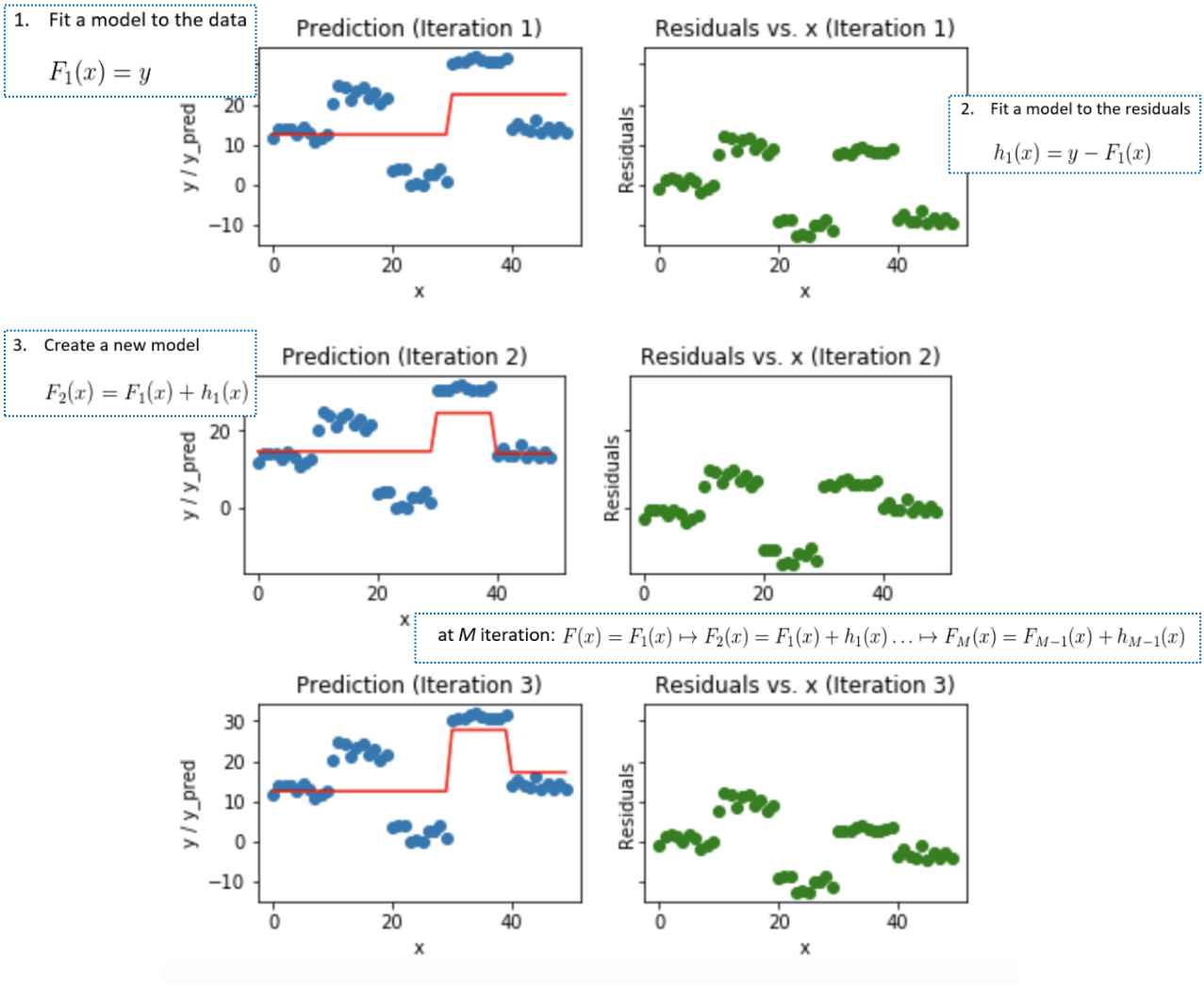

Figure 1.3.1: Visualization of gradient boosting predictions (First 4 iterations)[5]

[5] https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d

We first model data with simple models and analyze data for errors. These errors signify data points that are difficult to fit by a simple model. Then for later models, we particularly focus on those hard to fit data to get them right. In the end, we combine all the predictors by giving some weights to each predictor.

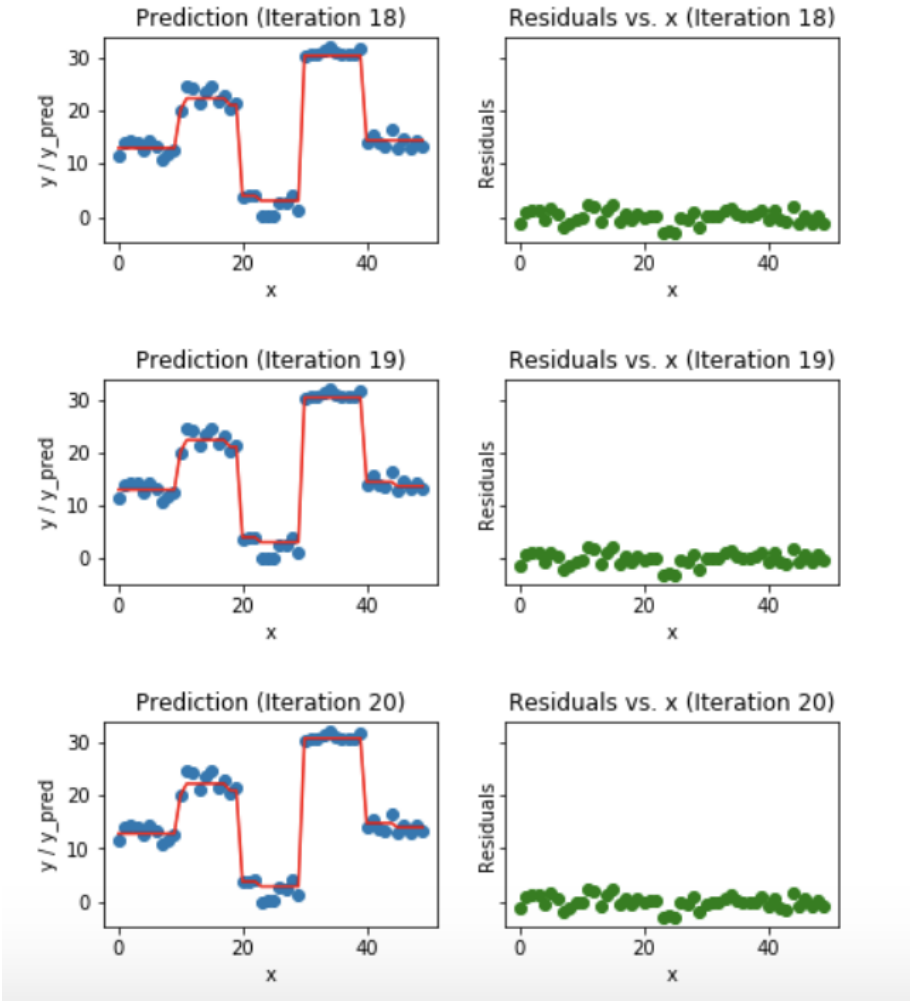

Figure 1.3.2: Visualization of gradient boosting predictions (18th to 20th iterations)

In figure 1.3.2, we observe that after 20th iterations, residuals are randomly distributed around 0 and our predictions are very close to true values, iterations are called n_estimators[6] in sklearn implementation. This would be a good point to stop training.

[6] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html

2 Benchmark the ML algorithms, interpret the results, explain the pros and cons of each ML algorithm

The data set is about condominium price in Bangkok, Thailand – obtained by webscraping. The target (dependent variable) is price per square meter in THB. After data cleaning and removing outliers process, there are 1,003 observations with 82 independent variables presented in the basetable. This was used as input for our ML models. Please refer to appendix for more information on data gathering, pre-processing, and hyperparameter tuning.

Model	RMSE	R-Squared
Linear (Baseline)	61,005.7	0.6694
Ridge	61,005.4	0.6694
RandomForestRegressor	32,458.3	0.6861
GradientBoostingRegressor	**29,012.3**	**0.7558**

Table 2.1: Result of the models on RMSE and R-Square metrics

Mean squared error is calculated by computing the square of all errors and averaging them over all observations. The lower this number is, the more accurate the predictions were.

R-Squared is the percentage of the observed variance from the mean that is explained by the model. It always falls between 0 and 1, and a higher number is better.[7]

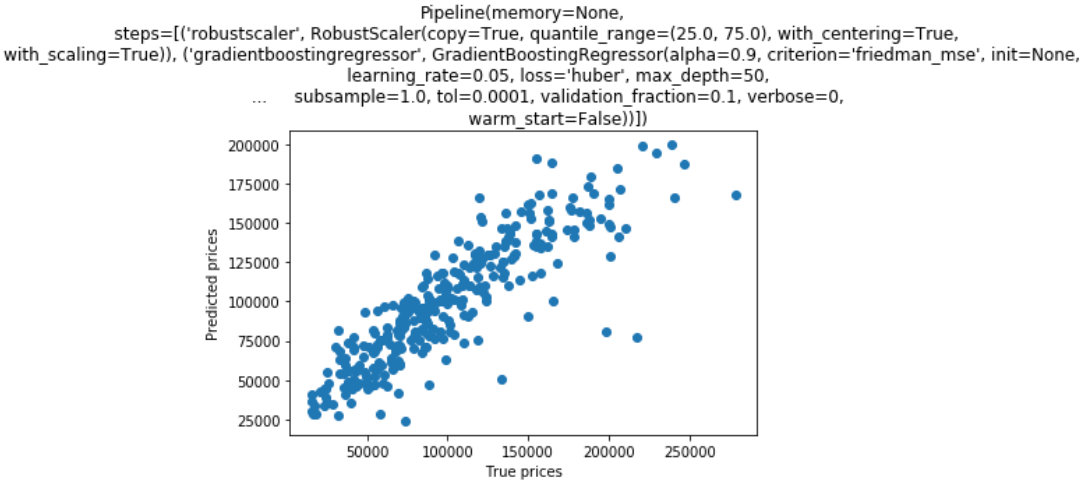

Figure 2.1: Scatter plot the result of Gradient Boosting Regressor

[7] https://pages.dataiku.com/machine-learning-basics-illustrated-guidebook

Table 2.1 shows the best result of each model after hyperparameter tuning process, in this project we used root mean square error (RMSE) and R-square as main matrix. The Gradient Boosting Regressor outperforms all the model with the lowest RMSE at 29,012.3 and the highest R-squared at 0.7558.

Figure 2.1 shows the scatter plot between actual price and predicted price resulted from gradient boosting.

The pros and cons for each model used in the project is explained below in table 2.2.

Model	Pros	Cons
Linear/ Ridge Regression	Fast to model and particularly useful when the relationship to be modeled is not complex, simple to understand which can be very valuable for business decisions.	These models are not as good as others when it comes to highly complex or non-linear data. Only regularization parameter to be tuned.
Random Forest Regressor	Great at learning complex, highly non-linear relationships. They usually can achieve quite high performance, better than polynomial regression. The decision boundaries that are built during training are easy and practical to understand.	Using larger random forest ensembles to achieve higher performance comes with the drawbacks of being slower and requiring more memory. The final trained model can learn complex relationships.
Gradient Boosting	Lots of flexibility with the choice of loss functions, adaptable to the characteristics of the studied problems. High performing.	Many parameters which can interact and influence heavily the behavior of the approach (number of iterations, regularization parameters, etc.) Overfitting can occur if values of parameters are not suitable. Computationally intensive especially when number of trees is high.

Table 2.2: Pros and Cons for each model

3 Summary and suggestion for future improvement

Even this dataset is quite small with lots of features and we can only predict the price per square meters for each condo, however, this study is very useful for buyers, resellers, agents and even developers to justify the 'fair price' as a starting point based on the current actual market data.

In the webscraping step, we should acquire all listings available in each condo, not only average price per sqm. This should increase numerous numbers of records and it would be very useful to estimate the price for every single room in the future.

We have scraped some quarterly historical prices but still did not use in this project since there were some unreliability issues in the data. It required some more detailed verification and data cleaning. This historical data can be really useful to visualize the trends for each condo/area (which areas are growing rapidly, which area are reaching plateau stage or declining).

Appendix

1. Data set and explanation webscraping process

This project uses Selenium library to firstly obtain all condominiums listed on the https://www.hipflat.com/ website, and extracts information for each page using BeautifulSoup package. Hipflat is one of the biggest property listing website in Thailand. This project is focused on condominium listings in Bangkok, both new and resale. Refer to below links for Python scripts.

001_Retrieve_all_urls.py

002_Scrape_info_for_each_condo.py

2. Pre-processing and data cleaning

Check NAs and data types for each column. Perform data manipulation by clean each column using regex, change numbers from strings to numeric, impute missing values, and convert lists of strings into columns. Refer to the link below.

003_data_cleaning_pred_current_price.py

3. Data scaling and hyperparameter tuning

Robust Scaler[8] is used in the pipeline before passing through the ML models. It uses a similar method to the Min-Max scaler but it instead uses the interquartile range, rather than the min-max, so that it is robust to outliers. It follows the formula[9]: $\frac{x_i - Q_1(x)}{Q_3(x) - Q_1(x)}$

[8] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html
[9] http://benalexkeen.com/feature-scaling-with-scikit-learn/

Hyperparameter tuning was done for each ML model as below:

Ridge (manually passing alpha through a list and for loop):

```
alpha_list =[0.0001,0.001,0.01,0.1,1,10,100,1000]
```

alpha	RMSE	R-squared
1000	55938.0	0.4165
100	59008.9	0.6030
10	61194.8	0.6297
1	61473.4	0.6489
0.1	60930.7	0.6635
0.01	60979.6	0.6687
0.001	61002.7	0.6693
0.0001	61005.4	0.6694

Random Forest (using `GridSearchCV`):

```
 param_grid = {
     'bootstrap': [True],
     'max_depth': [80, 90, 100, 110],
     'max_features': [9, 40],
     'min_samples_leaf': [3, 4, 5],
     'min_samples_split': [8, 10, 12],
     'n_estimators': [100, 200, 300, 1000]
 }
grid_search.best_params_

 #{'bootstrap':True,
 # 'max_depth': 80,
 # 'max_features': 40,
 # 'min_samples_leaf': 3,
 # 'min_samples_split': 8,
 # 'n_estimators': 200}
```

Gradient boosting (using `GridSearchCV`):

```
 param_grid = {
     'learning_rate': [0.05],
     'max_depth': [10, 20, 50, 100],
     'max_features': ['sqrt'],
     'min_samples_leaf': [3, 5, 10],
     'min_samples_split': [5, 10, 15],
     'n_estimators': [100, 200, 300, 1000,3000],
     'loss': ['huber'],
     'random_state': [121]
 }
grid_search.best_params_

 #{'learning_rate':0.05,
 # 'loss': 'huber',
 # 'max_depth': 50,
 # 'max_features': 'sqrt',
 # 'min_samples_leaf': 10,
 # 'min_samples_split': 5,
 # 'n_estimators': 3000,
 # 'random_state': 121}
```

For Random Forest and Gradient Boosting hyperparameter tuning, the tuning process took 11.4 and 62.8 minutes, respectively. Please refer to the script below.

004_1_ML_pred_current_price.py

For more information about the project and scripts, please refer to the github link below:

https://github.com/Ekapope/Predicting-condominium-price-using-data-from-webscraping