# Taiwan House Price Prediction

Data Mining Homework 1

B10704031 Po-Yen Chu

## 1 Abstract

This paper discusses the prediction of Taiwan house prices, including preprocessing methods, the chosen model, results, and evaluation. Before proceeding, note the execution details when running the provided notebook (.ipynb):

- **Environment**: The provided code should be run on Kaggle Notebook or Colab (with necessary file path modifications) to utilize CUDA (GPU) for improved efficiency.
- **Setup**: Two parameters can be adjusted: *target_y* and *layer_name*. *target_y* can be assigned either 'log_price' or 'Price', corresponding to the type of labels. *layer_name* can be assigned 1 or 2 (theoretically could be more, but the number of layers is randomly generated in this notebook); *layer_name = 1* indicates that the MLP model is not used.

## 2 Preprocessing

### 2.1 Feature Engineering

#### 2.1.1 Polar Coordinate

Location is a crucial feature for house prices. Initially, the price distribution among Latitude and Longitude is plotted, revealing concentration in the city center. Therefore, it is transformed into polar coordinates to establish a high correlation between price and rho.

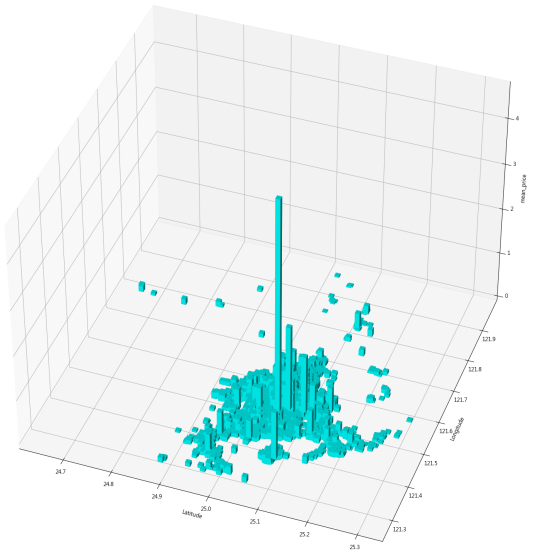

Figure 1: Price Distribution on Latitude & Longitude

#### 2.1.2 Feature Aggregation

Certain features require aggregation for improved continuity and representativeness:

- **Total_Area**: Computed as $total\_area = balcony\_area + aux\_building\_area + main\_building\_area$. Parking_Area is considered but not aggregated due to lack of correlation improvement.

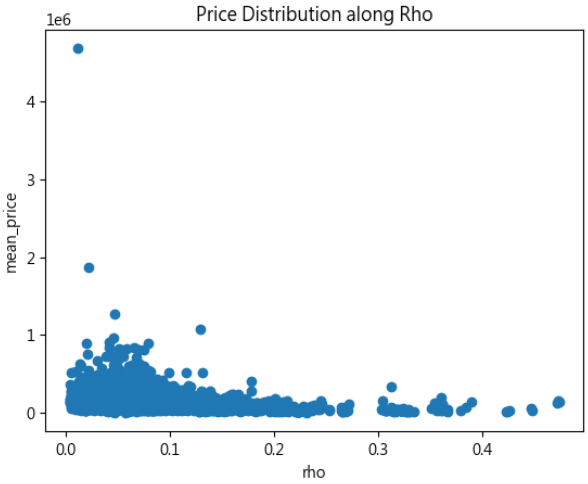


Figure 2: Price Distribution along Rho

- **Floor**: Determined by subtracting sell floor from total floor, resulting in a slightly higher correlation to price.
- **Total_Transfer_Area**: Computed as $total\_transfer\_area = land\_transfer\_area + building\_transfer\_area + parking\_transfer\_area$, with a slightly higher correlation to Price.

#### 2.1.3 Add Categorical Features

Observing 0 values in some continuous features infers their absence. Categorical features including *No_Balcony*, *No_Aux_Building*, *No_Built_Date* are added. However, features like *Parking_Area* with 0 values do not require additional features, as the original dataset adequately represents such differences.

#### 2.1.4 Remove Useless Features

Several duplicated or useless features are removed:

- **Deal / Built Date**: Deal dates show consistent year data with no significant patterns; hence, they are removed. Built date variations are preserved, with the year retained and the rest (month and day) discarded. Transforming dates to total days does not enhance model performance.
- **has_no_X**: To prevent collinearity problems, one of the features *has_X* and *has_no_X* is removed.

### 2.2 Outlier Removal

Given the importance of location, outliers are removed by grouping districts and discarding rows that exceed 1.5 standard deviations.

### 2.3 Normalization

#### 2.3.1 Y Transformation

Due to the severely skewed distribution of *Price*, it is transformed using *numpy.log1p()* and z-score normalization.


Figure 3: Comparison of Price Distribution

#### 2.3.2 X Transformation

All numeric columns are normalized to z-score. Mean and standard deviation are calculated only on train data to prevent data leakage; *X_test* data is normalized by the mean and std of *X_train*.

# 3 Models

## 3.1 Model Construction

This section discusses only the chosen submission models; other models not selected for submission are discussed separately.

### 3.1.1 Random Forest

As an ensemble learning method, Random Forest considers interactions between columns and is easier to train without domain knowledge.

## 3.2 Hyperparameters Tuning

### 3.2.1 Random Forest & XGBoost & CatBoost

Tuning methods for all models (except MLP) include Bayesian Optimization, implemented with *skopt.BayesSearchCV* for efficiency. Random Forest is particularly time-consuming, and due to limitations on GPU utilization, only *n_estimators* and *random_state* are controlled.

### 3.2.2 Multi-Layer Perceptron

With a more complex architecture and hyperparameters, random search is employed instead of Bayesian Optimization. Parameters such as batch size, learning rate, weight decay, and the numbers of layers and perceptrons are randomly generated within specified ranges.

# 4 Results Interpretation

## 4.1 Model Performance

Cross-validation ($cv = 3$) is used to evaluate training performance. The negative mean absolute error for the three folds respectively are *[-18564.4537336 -17809.81617093 -17707.85563999]*. While $cv = 10$ is empirically optimal, considering the time complexity of Random Forest, 3 folds are used.

## 4.2 Empirical Results

Apart from quantified comparisons, some decisions are based on empirical results, though statistics of performances are not provided.

### 4.2.1 Data Preprocessing

- **Y Transformation**: Validation scores suggest that *log_price* typically does not outperform *Price*, possibly due to the tree-based model's lesser sensitivity to skewed data.
- **Outlier Removal**: While outlier removal reduces training loss, it also increases overfitting tendencies. Consequently, outlier removal is not executed in the provided code.

### 4.2.2 Model Selection

- **Multi-layer Perceptron**: Despite successful application in previous projects, MLP's performance is subpar even after tuning, possibly requiring better preprocessing or more data.
- **XGBoost**: While XGBoost performs well as an ensemble learning model, it does not outperform Random Forest.

- **CatBoost**: Despite its efficacy in handling categorical data, CatBoost does not perform as expected.
- **MLP + Random Forest**: Inspired by previous research, the combination of MLP and Random Forest did not yield satisfactory results, possibly due to interactions between columns not being properly analyzed by Random Forest after feature extraction.

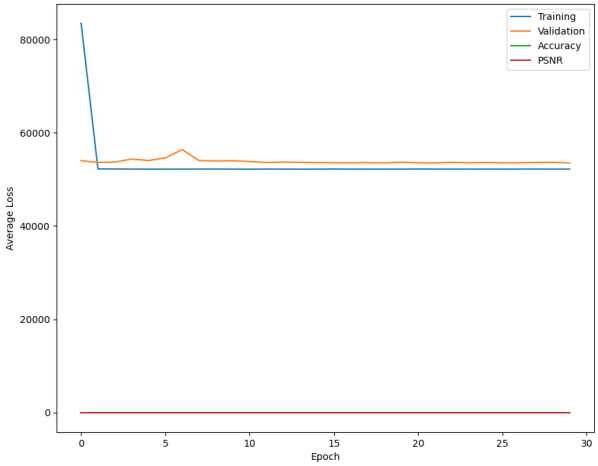


Figure 4: MLP Training Loss over epoch

## 4.3 Conclusion

As we conclude, there are several promising avenues for future research to pursue.

Firstly, exploring techniques to dynamically adjust computational resources, such as leveraging *cuml* in appropriate environments to accelerate algorithms like Random Forest, holds significant potential for enhancing overall efficiency further.

Secondly, investigating the reasons for failures in MLP structures, which can consume considerable development time, warrants deeper examination, particularly focusing on preprocessing methodologies to address potential bottlenecks.

Lastly, there's a compelling need for more extensive feature engineering, especially in extracting location-based features like mean prices from different districts, considering the intrinsic relationship between such features and house prices. These endeavors promise to enrich our understanding and predictive capabilities in the domain of housing price prediction.

# References

[1] Kumar, S., Bhatt, S., Phudinawala, H. (2023) *Predicting House Price with Deep learning: A comparative study of Machine Learning Models.*