EE4802 - Learning From Data - House Price Prediction Project Report

4th March 2023

Lorcan Nicholls
National University of Singapore

Contents

1. Abstract

A regression-based predictor for HDB resale prices in Singapore is presented, having been trained on a large dataset provided by a government database. The design choices in constructing the regression model are outlined and justified in this report, and its accuracy is examined under a K-fold cross-validation scheme. It was found that the model performed well, with a root mean squared error (RMSE) of under $60,000, representing a mean absolute percentage error (MAPE) of approximately 10%. A web application is also designed, which is suitable for usage by the general public and could readily be embedded into a larger service.

Keywords: machine learning, regression, Python, scikit-learn

References: In this report, the following resources are referenced.[1]

[1] *HDB Flat Prices*, Singapore Government Dataset (2023)
https://data.gov.sg/dataset/resale-flat-prices

[2] *OneMapSG*, Singapore Government Website (2023)
https://www.onemap.gov.sg/docs/

[3] *PCA and SVD explained with NumPy*, Z. Wang (2019).
https://towardsdatascience.com/pca-and-svd-explained-with-numpy-5d13b0d2a4d8

[4] *House Price Prediction*, Python Anywhere, L. Nicholls (2023)
https://lorcan2440.pythonanywhere.com/

[5] *House Price Prediction*, Personal Website, L. Nicholls (2023)
https://lorcan.netlify.app/project/house-price-prediction/

[1] Note to Professor: The references are placed here in order to fit the imposed page limit.

2. Data Acquisition, Exploration and Preprocessing

The initial raw dataset was obtained from the Singapore government's public datasets[1], using an API to fetch and download data on flat resale prices. Data from 2012 to present was used, since prior to this date, alternative fields were used which could result in a bias due to the change in data collection methodology. The incoming data was saved locally as a JSON file with 234,235 records of 11 variables totalling 44 MB in memory.

The data was loaded in Python. All data manipulation was performed using the `pandas` and `NumPy` libraries. The labels `remaining_lease` and `storey_range` were deemed unsuitable for the prediction of resale price, and were dropped from the dataset. The labels `town` and `street_name` were used to find the latitude and longitude of the house using the OneMapSG API[2]. These coordinates were used to generate a map using the `plotly` library, shown in **Fig. 1**. An analytics report of this dataset was then generated using `pandas-profiling`, and analysed for potential correlations. Observations of the map data showed clusters but no clear geographic trend in house price was evident.

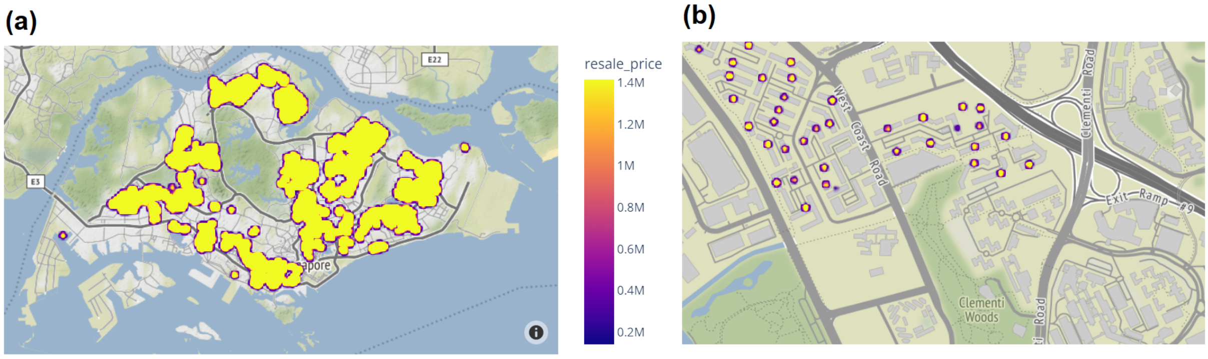


Fig. 1. Map of recorded house locations, displayed as a `Plotly` density heatmap.
(a) Whole of Singapore; **(b)** Flats in Clementi, showing the high precision of the data.

The remaining labels were processed as follows:

- `flat_type`: ordinally encoded from 0 to 7 (1-5 rooms < Executive < Multi Generation).
- `flat_model:` ordinally encoded using the mean `resale_price` grouped on `flat_model`.
- `town`: ordinally encoded using the mean `resale_price` grouped on `town`.
- `month`: parsed using `datetime` and converted to a number of months since January 2012, and used to calculate `age_years` (years between the lease start date and sell date.

Contrary to expectations, it was found that the one-hot encoding of `town` produced poor results during preliminary testing. Therefore, the grouping method was used to rank towns by their mean `resale_price` value. This provided 9 features to use in our analysis, which were `floor_area_sqm`, `lease_commencement_date`, `lat`, `long`, `flat_type_rank`, `flat_model_rank`, `town_rank`, `months_since_2012` and `age_years`, all of which were numeric. By design, `lease_commencement_date`, `months_since_2012` and `age_years` would be mutually dependent, but it was decided to keep them as correlations would be excluded in a subsequent processing step (principal component analysis).

3. Model Selection and Hyperparameter Tuning

The preprocessed dataset with 9 features was transformed in a variety of ways and tested in cross-validation with 10 splits in an 80:20 train-test size ratio, as shown in **Table 1**, from which the best performing model was selected. All polynomial features were normalised to zero mean and unit variance prior to regression, as required by principal component analysis (PCA). Where principal component regression (PCR) was used, all components were included initially, and no regularisation was applied. For a rigorous review, see the Appendix.

Poly Degree	Model Type	Root Mean Squared Error (RMSE)
1 (linear) (9 features)	Linear Regression	90,590
	Lasso Regression	90,600
	Ridge Regression	90,590
	Principal Component Regression	152,800
2 (quadratic) (54 features)	Linear Regression	66,270
	Lasso Regression	72,190
	Ridge Regression	67,670
	Principal Component Regression	64,970
3 (cubic) (219 features)	Lasso Regression	66,680
	Ridge Regression	59,510
	Principal Component Regression	54,680

Table 1. Model performance using a variety of dataset transformations and regressors.
The 3rd degree linear regressor did not yield a valid model, as it returned an extremely large RMSE and is omitted.

In order to assess the potential for model dimensionality reduction, plots were made of the RMSE and variance proportion against the number of principal components (PCs) used.

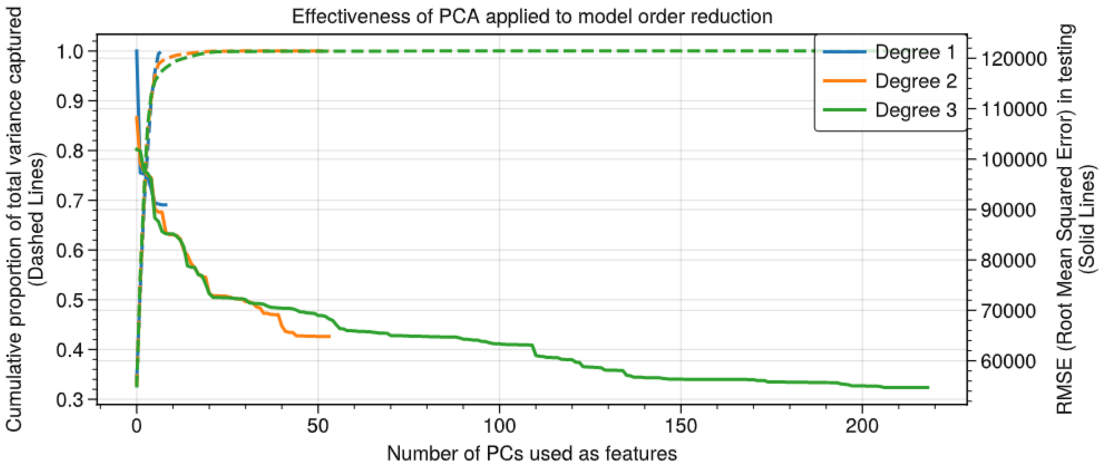


Fig. 2. Scree plot and error plot for degree 1, 2 and 3 polynomial fits after PCA.

It was decided to use the first 136 PCs (a 40% reduction in the dimension count), which gave an RMSE of 57,600 (5% higher than using all PCs), while using 100% of the variance.

4. Model Prediction and User Interfacing

Fig. 3 summarises the data processing stages used by the program to make predictions.

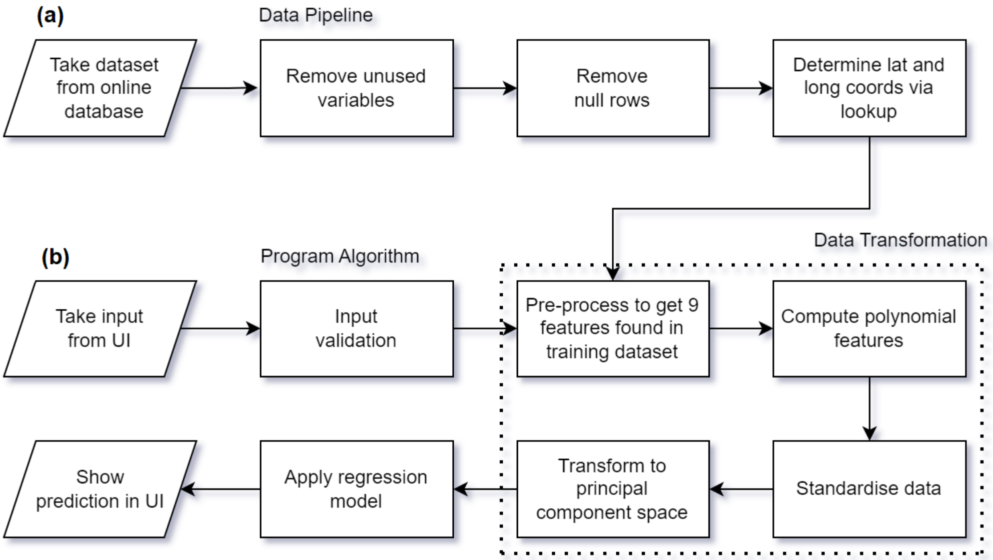
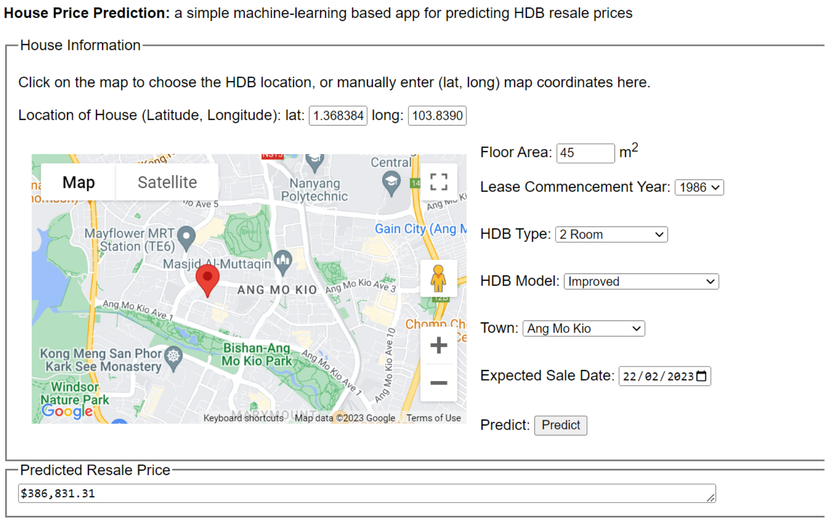

(a)
Data Pipeline
Take dataset from online database
Remove unused variables
Remove null rows
Determine lat and long coords via lookup
(b)
Program Algorithm
Data Transformation
Take input from UI
Input validation
Pre-process to get 9 features found in training dataset
Compute polynomial features
Standardise data
Transform to principal component space
Apply regression model
Show prediction in UI

Fig 3. A flow diagram showing the main steps in **(a)** training the model, and **(b)** making a prediction given a user's input.

A simple web application was written using Flask. Various design considerations were taken into account in order to improve the accessibility and ease of use by the general public:

- A responsive layout was used, which fits screen dimensions on any platform.
- Users can choose the house location by clicking on an interactive map, which drops a pin and automatically fills in the coordinates, provided via the Google Maps JS API.
- Familiar text fields with example default values and buttons are used to handle user input selection. Inputs are validated by exception handling and shown with an error.
- The app was deployed on the Cloud service PythonAnywhere[4] and embedded in my personal website[5], as a demonstration of its ease of integration into larger services.

Fig. 4 shows a screenshot of the web application in use, with a prediction shown.


House Prediction: a simple machine-learning based app for predicting HDB resale prices

House Information

Click on the map to choose the HDB location, or manually enter (lat, long) map coordinates here.

Location of House (Latitude, Longitude): lat: 1.368384 long: 103.8390

Floor Area: 45 m^2

Lease Commencement Year: 1986

HDB Type: 2 Room

HDB Model: Improved

Town: Ang Mo Kio

Expected Sale Date: 22/02/2023

Predict: Predict

Predicted Resale Price

$386,831.31

Fig. 4. Web app screenshot, showing the program in use, after a prediction has been made.

5. Appendix

Appendix 1: Theory of Linear Regression

A regressor aims to find unknown coefficients of a mathematical model such that a certain loss function is minimised. The loss function is typically the sum of two terms: a penalty on the model's inaccuracies, and a penalty on the model's choice of coefficients. The former term typically represents a least-squares problem, but in some cases (e.g. support vector machines), the absolute error is used. The latter term is known as regularisation, and aims to reduce the magnitude of the coefficients in order to avoid overfitting. Various types of regularisation exist e.g. Lasso (L_1 norm) and Ridge (L_2 norm). The model objective can be stated as an (in general, nonlinear and not convex) optimisation problem (where x is a set of features). For example, the ridge regressor is characterised by:

Find $\{w_0, w_1, \ldots, w_M\}$ which minimises $\displaystyle \frac{1}{2N}\sum_{i=1}^{N}(y_i-\hat{y}_i)^2+\alpha\sum_{j=1}^{M}|w_j|$ with $\displaystyle \hat{y}_i=\sum_{j=0}^{M}w_jx_{ij}.$

(y_i: observed target variable, $\hat{y}_i$: predicted target variable, N: number of training data points, M: number of independent variables to be fitted, α: regularisation hyperparameter, $x_{i,0}$:= 1)

If polynomial regression is used, then x contains all product combination terms up to a specified degree. This significantly increases the number of features used, many of which are strongly correlated.

Appendix 2: Theory of Principal Component Analysis

Principal component analysis (PCA) identifies a set of new variables which are linear combinations of the existing features in a dataset. PCA aims to produce variables which are the most uncorrelated with each other, and thus are useful for model order reduction. Considering a dataset of n predictor variables (features) with m observations, the data matrix $\mathbf{X}$ is defined as the $m \times n$ matrix containing each observation in a row, where all columns (features) are rescaled to zero mean. If the singular value decomposition (SVD) of $\mathbf{X}$ is denoted as $\mathbf{X} = \mathbf{Q}_1\mathbf{\Sigma}\mathbf{Q}_2^{\mathrm{T}}$, then the columns of $\mathbf{Q}_2$ (rows of $\mathbf{Q}_2^{\mathrm{T}}$) are the principal components (PCs). The PCs can be interpreted as the principal axes of the m-dimensional ellipsoidal contour surfaces of equal probability density for the dataset. The singular values σ, listed in descending order along the leading diagonal of $\mathbf{\Sigma}$, represent the relative significance of each PC. The matrix $\mathbf{Q}_1\mathbf{\Sigma} = \mathbf{X}\mathbf{Q}_2$ is the transformed dataset, whose features are guaranteed to be independent, since $\mathbf{Q}_2\mathbf{Q}_2^{\mathrm{T}}$ is a symmetric square (real Hermitian) matrix and thus has a full set of orthogonal right eigenvectors.

An alternative computation uses the eigendecomposition of the covariance matrix $\mathbf{C}$ where $\mathbf{C} = (m - 1)^{-1}\,\mathbf{X}\mathbf{X}^{\mathrm{T}}$, so that $\mathbf{C} = (m - 1)^{-1}\,\mathbf{Q}_2\mathbf{\Sigma}^2\mathbf{Q}_2^{\mathrm{T}}$. The eigenvalues of $\mathbf{C}$, given by $\lambda = (m - 1)^{-1}\,\sigma^2$, then represent the variances of data in the respective PC, which can be plotted to form a 'scree plot'. This is useful for choosing the desired number $k \leq n$ of PCs to retain such that the most variance is captured, while reducing the model dimension count and the memory space required. However, in Python's `scikit-learn` library, the SVD approach is used, since highly optimised algorithms exist for the efficient computation of a direct SVD[3].