Property Sale Price Prediction in the city of Daegu

Edward Bickerton
Department of Computer Science
University of Bristol
rw19842@bristol.ac.uk

***Abstract*—This report was completed as coursework for the module Introduction to Artificial Intelligence (EMATM0044). In this report, to make predictions on property sale prices I apply two supervised learning techniques; decision tree regression and a multi layer perceptron (MLP), detailing the methods used for selecting hyperparameters and comparing their performance to a baseline model.**

I. Introduction

A. Dataset

The dataset, `coursework_fintech.csv`, contains information about property sold in the city of Daegu including sale price, year and month sold along with various features of the property such as size and year built. Since we are trying to predict sale price, a continuous variable, regression algorithms are the natural fit for this problem.

All features are numeric except for `AptManageType` and `SubwayStation` which are strings, the models cannot directly handle this and so some pre-processing is required to convert these strings to integers. Since `AptManageType` takes values `self_management` and `management_in_trust` I use binary encoding, replacing `self_management` with 1 and `management_in_trust` with 0. For `SubwayStation` which is categorical data, I use one-hot encoding [1] which converts the single column into a column for each subway station, a 1 in a station's column signifies it being the nearest station to the property otherwise it takes the value 0.

Making sure to shuffle the data beforehand, I create a train test split using 80% of the data for training and the remaining is kept aside as test data. I write these datasets into new csv files so that they can be used across the models.

B. Performance Metric

The two most commonly used metrics for evaluating the performance of regression models are mean squared error (MSE) and mean absolute error (MAE). I choose MAE for the following reasons: being in the same units as the sale price I find MAE to be more natural and interpretable than MSE, MAE is less sensitive to outliers in the error [2] and given the context of the task an error of £20,000 should be treated as twice as bad as an error of £10,000 (and not four times, as MSE would).

II. Baseline Model

To serve as a baseline of performance I create a dummy model using `DummyRegressor` from the scikit-learn module `sklearn.dummy`. It is a very simple regression model, outputting a constant function equal to either the mean or median of the training data. Fig. 1 shows the distribution of sale prices in the training data.

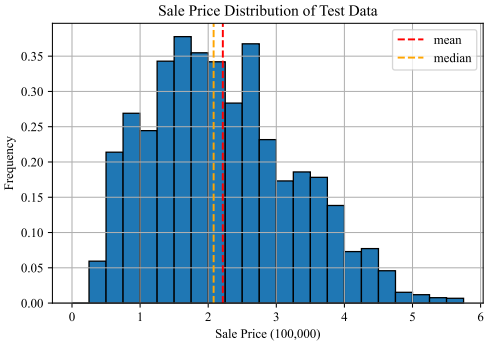


Fig. 1: Distribution of sale prices in the training data.

Since there is a positive skew to the sale price (and since the dummy model needs all the help it can get) I'll use the median for my dummy model as the median offers a (slightly) better measure of centrality.

III. Decision Tree

I use scikit-learn's `DecisionTreeRegressor` from the `sklearn.tree` module [3]. Unfortunately, scikit-learn's implementation of a decision tree is much slower when using MAE to evaluate the quality of split than when it uses MSE [4], for this reason I use MSE as criterion for picking splits.

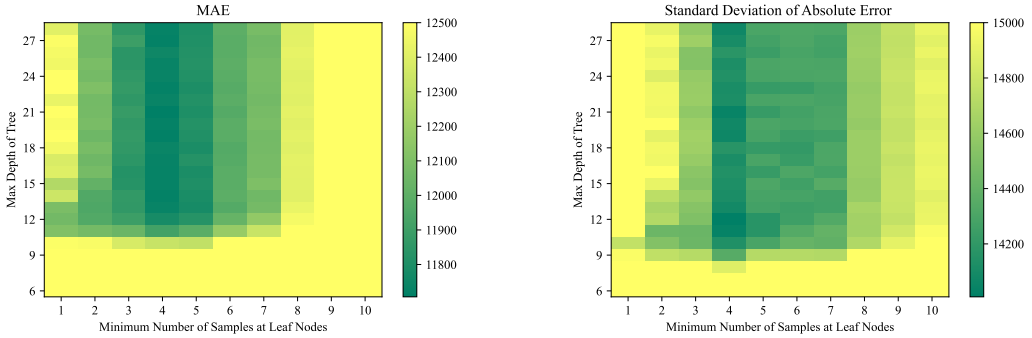


Fig. 2: Heatmap showing performances of decision trees using different amounts of pruning.

To avoid overfitting, I use parameters `max_depth` to control the maximum depth of the tree and `min_samples_leaf` which controls the minimum number of samples at leaf nodes (in effect pruning the tree).

A. Hyperparameter Selection

To select optimal values for `max_depth` and `min_samples_leaf` I train a decision tree for each pair of hyperparameters in $[6, 28] \times [1, 10] \subset \mathbb{Z}^2$ (Note: the depth of the tree without pruning is 28). Since decision trees can be unstable, I use cross validation, splitting the training set into 10 folds and cycling through, training a decision tree using one fold for validation and the remaining nine as training data. Using the validation dataset, I calculate the performance of the tree using the MAE and, since decision trees are prone to instability, I take the standard deviation of the absolute error. Finally, I take the mean of these statistics across folds, fig. 2 shows the results.

I take the top 14 (there is a sharp increase in MAE after 14) models with smallest (average across folds) MAE and from these choose the model which had the smallest standard deviation of absolute error. This gives optimal hyperparameters 21 for `max_depth` and 4 for `min_samples_leaf`.

IV. Multi Layer Perceptron

For the MLP I chose PyTorch over scikit-learn for GPU support [5], using [6] for the basis of my own implementation.

Because the MLP is sensitive to the scale of features [7], the model requires additional data pre-processing to scale the values between 0 and 1. For this I fit scalers from `sklearn.preprocessing` on the training data before transforming both the train and test data.

I use mini batch gradient descent [8], which updates the neural network (NN) weights based on the average gradient (with respect to said weights) of the loss function (MAE) taken from a batch of the training data. A training iteration (or epoch) is a single pass through the entire training set. Thus, if the batch size is smaller the NN weights are updated more frequently (which increases the time taken to complete an epoch), whereas for larger batch sizes the NN weights are updated less frequently but, being the average of a larger number of samples, the gradient more reliably points towards the minimum loss.

A. Neural Network Architecture

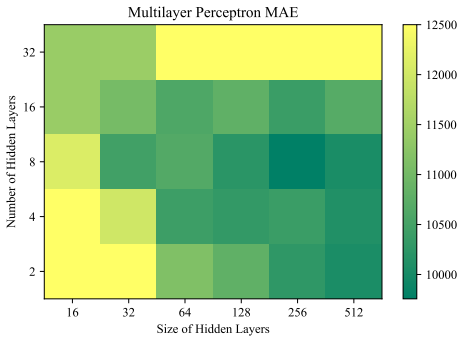
To keep the number of hyperparameters to a minimum, each hidden layer is fully connected and of the same size (in any case it was shown in [9] that using the same size for all layers worked generally better or the same as decreasing or increasing the sizes). To introduce non-linearity into the model I use $\texttt{ReLU}(x) = \max(0,\ x)$ in-between each of the layers as opposed to the sigmoid function as it is less computationally expensive and doesn't suffer from the vanishing gradient problem [10].

This leaves the following hyperparameters to tune: number of hidden layers, size of hidden layers, number of training iterations (epochs), batch size and learning rate.

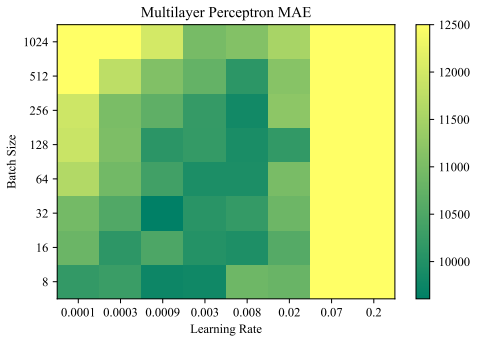
B. Hyperparameter Selection

Since training a NN is much more computationally expensive than training the decision tree it is not feasible to train hundreds of models required for cross validation. So, for the MLP I will simply split the training data further to create a train-validation-test split, using 60% of the data for training and the remaining 40% is split equally between test and validation data. Also, to reduce the size of the search space of the hyperparameters I will implement several ideas from [11];

- number of epochs: I use early stopping i.e., training the model for a larger number of epochs than required (namely 500) then looking at the learning curve and selecting a value for which the model's performance on validation data has converged. Since early stopping is the strongest mechanism for preventing overfitting, "it hides the overfitting



(a) Selecting hidden layer size and number of hidden layers, using default values of learning rate and batch size without early stopping.



(b) Selecting learning rate and batch size using the optimised values for hidden layer size and number of hidden layers, still without early stopping.

Fig. 3: Heatmap showing performances of MLP using different values of hyperparameters.

effect of other hyperparameters" (as stated in section 3.1.1 of [11]). Thus, early stopping will only be used once the other hyperparameters have been selected.

- Batch size and learning rate: since batch size mainly effects training time and not so much training performance and since batch size and learning rate may interact, I will optimise number of hidden layers and size of hidden layers using a default value of 32 for batch size (as recommended in [12]) and 0.01 for learning rate, before optimising batch size and learning rate together as recommended in section 3.1.1 of [11].
- Scale of values considered: to test a wide range of values of hyperparameters without needlessly trying out many candidate hyperparameters, I take a uniform sampling in the log-domain of each hyperparameter as suggested in section 3.3.1 of [11].

After training 94 models fig. 3 shows the results of the hyperparameter selection. A MLP with 8 hidden layers of size 256 using a learning rate of 9×10^{-4} and batch size of 32 is selected (for having the lowest MAE). All that remains is to reduce the number of training iterations to avoid overfitting.

From fig. 4 we can see that the performance of the model converges after about 150 epochs, beyond which additional epochs will only result in an unstable and overfit model. Thus, I set the hyperparameter to 150.

V. Results and Analysis

Once the hyperparameters have been selected I re-train both models using the entire training set, their performances on the test data can be seen in table I. Both models significantly outperform the baseline and table I suggests that the Multilayer Perceptron out performs the Decision Tree.

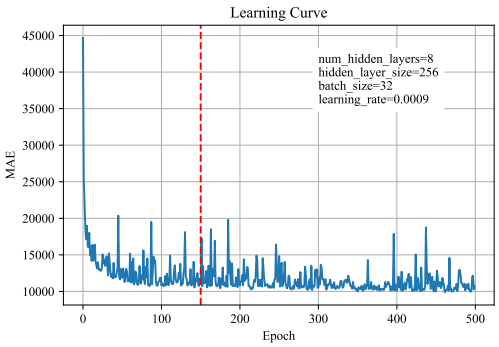


Fig. 4: Performance of the MLP using optimised hyperparameters against the number of epochs.

TABLE I: Model Performances

	MAE
Baseline	88,212.58
Decision Tree	11,648.72
Multilayer Perceptron	10,472.80

To make the conclusion that the MLP outperforms the decision tree statistically rigorous, I use cross validation. Splitting the entire data set into 10 folds and training 10 decision trees and multilayer perceptrons, using a different fold each time as the test data. From this I obtain 10 values of the MAE for each model which can be seen in fig. 5. The two box plots hardly overlap supporting the conclusion and a paired t-test gives a p-value of 3.33×10^{-5} confirming the conclusion further.

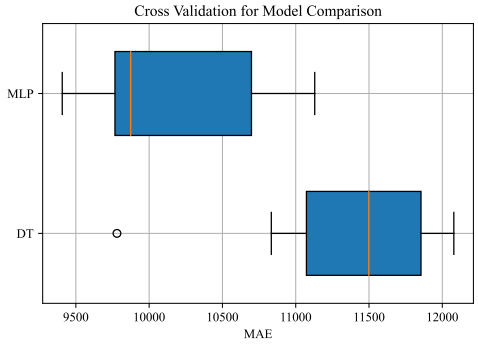


Fig. 5: Performance of the multilayer perceptron and decision tree using 10-fold cross validation.

VI. Conclusion

Although the MLP outperforms the decision tree, there are other things to consider when deciding between the two models.

The decision tree does not require the same level of pre-processing as the MLP and the MLP required tuning more hyperparameters. Decision trees can handle both numeric and categorical data and even handle missing values (although scikit-learn's implementation cannot [13]).

Decision trees are highly interpretable. For example, table II shows the top five most important features for creating splits in the decision tree, this could be valuable when deciding what information is key when marketing a house. Plotting the tree as in fig. 6, allows you to understand *why* a specific house was given a particular prediction of sale price, and could aid a developer in making the most cost-effective improvements to a property to maximise return on investment.

Overall, the decision between which model to use will depend greatly on the specifics of the application, including the client's needs.

TABLE II: Feature Importance

Feature	Relative Importance
Size(sqf)	0.56
YrSold	0.19
N_FacilitiesNearBy(ETC)	0.14
N_Parkinglot(Basement)	0.05
YearBuilt	0.02

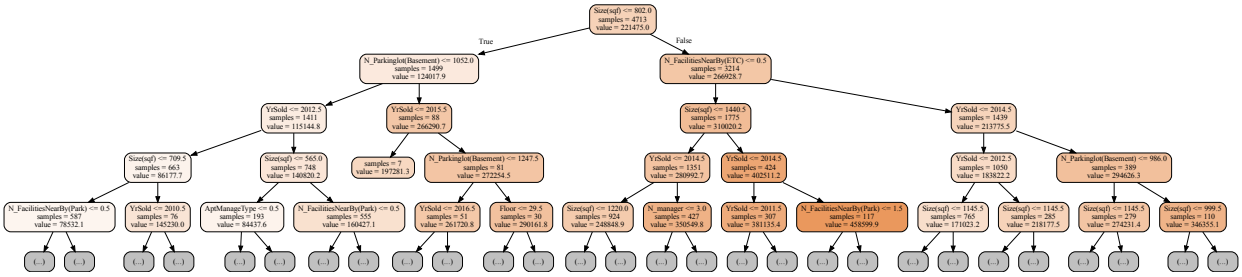


Fig. 6: A graph of the decision tree cropped to a depth of four. The colour of the nodes represents the sale prices of the samples at the node, darker being properties with a higher sale price.

References

[1] Jason Brownlee. "Why One-Hot Encode Data in Machine Learning?" In: *Machine Learning Mastery* (June 2020). URL: https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/.

[2] Cort J. Willmott and Kenji Matsuura. "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance". In: *Climate Research* 30.1 (2005), pp. 79–82.

[3] scikit-learn. *sklearn.tree.DecisionTreeRegressor*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html.

[4] GitHub user mcgibbon. "Use median instead of mean when constructing RandomForestRegressor". In: *scikit-learn GitHub issue #9553* (2017). URL: https://github.com/scikit-learn/scikit-learn/issues/9553#issuecomment-322330522.

[5] "Will you add GPU support?" In: *scikit-learn.org FAQ* (2023). URL: https://scikit-learn.org/stable/faq.html#will-you-add-gpu-support.

[6] Adrian Tam. "Building a Regression Model in PyTorch". In: *Machine Learning Mastery* (Apr. 2023). URL: https://machinelearningmastery.com/building-a-regression-model-in-pytorch/.

[7] scikit-learn. *1.17.8. Tips on Practical Use*. URL: https://scikit-learn.org/stable/modules/neural_networks_supervised.html#tips-on-practical-use.

[8] Jason Brownlee. "A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size". In: *Machine Learning Mastery* (Aug. 2019). URL: machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/.

[9] Hugo Larochelle et al. "Exploring Strategies for Training Deep Neural Networks". In: *J. Mach. Learn. Res.* 10 (June 2009), pp. 1–40. ISSN: 1532-4435.

[10] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks". In: *International Conference on Artificial Intelligence and Statistics* (2011).

[11] Yoshua Bengio. "Practical recommendations for gradient-based training of deep architectures". In: *arXiv* (2012).

[12] Dominic Masters and Carlo Luschi. "Revisiting Small Batch Training for Deep Neural Networks". In: (2018).

[13] scikit-learn. *Decision Trees*. URL: https://scikit-learn.org/stable/modules/tree.html#tree.