COMP60012 - House Value Regression Project Report

Steven Chen <shc20> Thomas Wong <tyw21> Rob Wakefield <rgw20> Tsz Chun Jason Wong <tcw121>

1 Regression Model

1.1 Preprocessor

The input training and test data is passed through the preprocessor. Missing data entries in the dataset are filled in with a default value using *fillna*. The categorical data in the *ocean_proximity* column is split into 5 columns for the 5 possible entries using *LabelBinarizer*. Finally, the data is min-max normalised based on training data only. Shuffling of the data is done before the preprocessor by the *test_train_split* function which splits training data with the final test data. This ensures that the model is never trained on the final test data.

1.2 Model

After the data has been preprocessed, there are 13 inputs to the model and 1 output (the median house value). The number of hidden layers and the number of nodes in each hidden layer are hyperparameters that can be passed into the constructor as a list of integers. Between each layer, there is a ReLU() activation function. A linear activation function is used on the output node as the median house value is unbounded. The weights are initialised using Xavier Glorot and the biases are initialised to zero.

1.3 Training

The loss function used is Mean Squared Error along with the Adam gradient descent function which utilises an adaptive learning rate. The initial learning rate passed to the Adam function is still used as a hyperparameter.

After the *fit* function has been called and the training data has been preprocessed, the data is split into batches of an equal size that can be passed to the *Regressor* constructor. During each epoch, mini-batch gradient descent is performed on the batches in order to achieve good training performance and to reduce the effect of noise in the training data.

2 Evaluation Setup

The dataset is shuffled and split into a training set and a test set using the 90% and 10% ratio of the total dataset respectively. All training and optimal hyperparameters searching only uses the training set, which ensures no model fitting is done on the testing set to avoid over-fitting. An optimisation of the model would be to do cross-validation to get an accurate result, however, the current implementation already takes at least an hour to run and since performing a 5-fold cross-validation would take additional hours, the current implementation is proposed to be of sufficient accuracy.

We made use of Mean Squared Error to measure the performance of the model during training and the hyperparameter search.

3 Hyperparameter Tuning

3.1 Implementation

We used *GridSearchCV* from scikit-learn to exhaustively search over all the hyperparameter values. Cross-validation (5-fold) is implemented to avoid overfitting when evaluating the scores against the validation set with negative mean squared error. The evaluation is done in parallel to speed up the tuning time

As an *estimator* object needs to be provided for scikit-learn's *GridSearchCV*, we made an adaptor, *RegressorAdaptor* for our implementation of *Regressor* to use in the grid search.

3.2 Choosing hyperparameter candidates

These are the values we picked by trial and error along with the reasons stated.

- Learning Rate: 0.1, 1, 10

  Reason: We selected a range that covers a spectrum from small to large learning rates. Since we are using adaptive learning rate gradient descent the initial learning rate doesn't have a huge impact, as such we only test a relatively small set.

- Hidden layers sizes: [8], [16], [32], [64], [16, 8], [32, 16], [64, 32], [64, 32, 16]

  Reason: We chose a wide range of hidden layer architectures up to three layers as studies have shown three layers are usually enough[1]. We picked the number of neurons in each layer with powers of 2 as it is the convention and it might have a slight computation advantage at the hardware level.

- Epoch: 10, 100, 1000, 10000

  Reason: We have selected a wide range of epochs, as too large or too small would suffer from over-fitting and under-fitting respectively.

- Batch size: -1 (No batching), 5000, 1000 500

  Reason: We chose batch sizes ranging from the dataset size to 500. These values were selected based on the input dataset size - they roughly approximate to 1/2, 1/10 and 1/20 the size of the total training dataset.

3.3 Evaluation and Findings

3.3.1 Evaluation

The scores of all combinations are extracted to CSV files by the function *printHyperParamsScores*, which allows us to present our findings below.

Rank	Batch size	Hidden layers sizes	Learning rate	Epoch	Score
1	500	[64, 32]	0.1	10000	-2911543350
2	1000	[64, 32]	0.1	10000	-3011921287
3	500	[32, 16]	1	10000	-3035262941
4	1000	[32, 16]	1	10000	-3070933288
5	1000	[64, 32]	1	1000	-3376101830

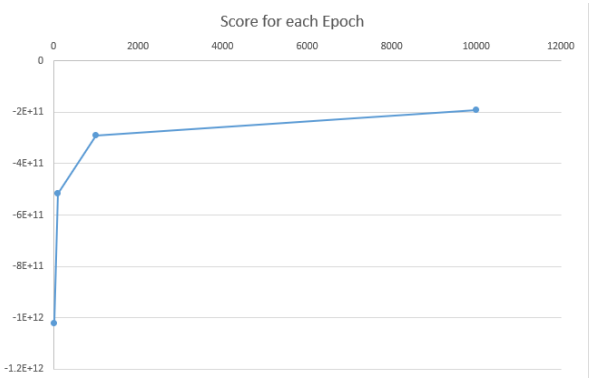
Table 1: Top 5 scoring combinations of hyperparameters

The scores computed are the mean performances of each combination of hyperparameters after 5-fold cross-validation. It is negative since our grid search is scored with negative mean square error so a higher score corresponds to a higher accuracy model. Table 1 shows the top 5 combinations of hyperparameters in the grid search. It suggests that the best 5 models have at least 2 layers, an epoch of mostly 10000 and a batch size of less than 1000. Our hyperparameter tuning method picks the best-performing hyperparameters out of all possible combinations i.e. batch size=500, hidden layers sizes=[64, 32], learning rate=1, epoch=10000.

3.3.2 Findings

We are able to isolate the impact caused by the changes of one hyperparameter p by plotting performance against the change of one hyperparameter. We get the scores of one hyperparameter by summing the scores of the models with the combination of hyperparameters where they have the same value v for hyperparameter p. In other words, the score of a value of one hyperparameter is defined as $Score_p(v) = \sum_c^{C_p(v)} Score_c$ where $C_p(v)$ is all the combination where they have the same value v for hyperparameter p.

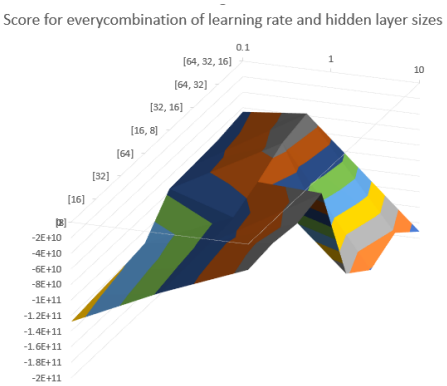
(a) Performance against change of learning rate



(b) Performance against change of epoch

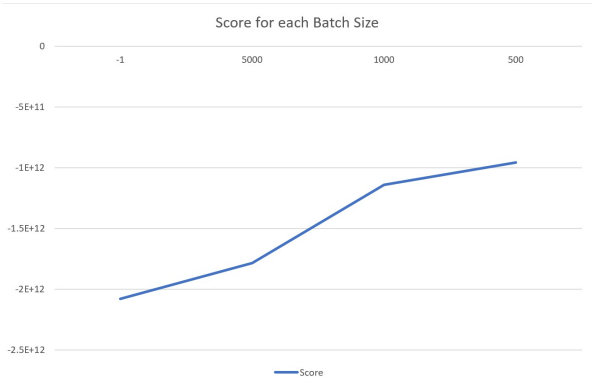
Figure 1a shows that the performance of the model peaks when the learning rate equals 1 but drops sharply at 10 as it is too large causing the model to overshoot the minimum. For the number of training epochs, the performance increases indefinitely and peaks at 10000 (Figure 1b). Since the performance between 1000 and 10000 is not significant, we could pick 1000 as the best parameter to decrease training time, but it is not implemented in our current model.

(a) Performance against change of hidden layer sizes (ordered by number of layers)



(b) Performance against combinations of learning rate and hidden layer sizes

Figure 2a shows that the performance of the model peaks at the layer configuration [64], but drops sharply as the model size increases. This information is misleading, as a larger hidden layer might require a higher learning rate. Figure 2b shows that the performance peaks at learning rate = 1 for multi-models and learning rate = 10 for single-layer models with similar scores. The worse performance for larger models suggested by Figure 2a may be caused by the bad performance of larger models with a learning rate of 0.1 or 10 due to not reaching and overshooting the minima respectively.



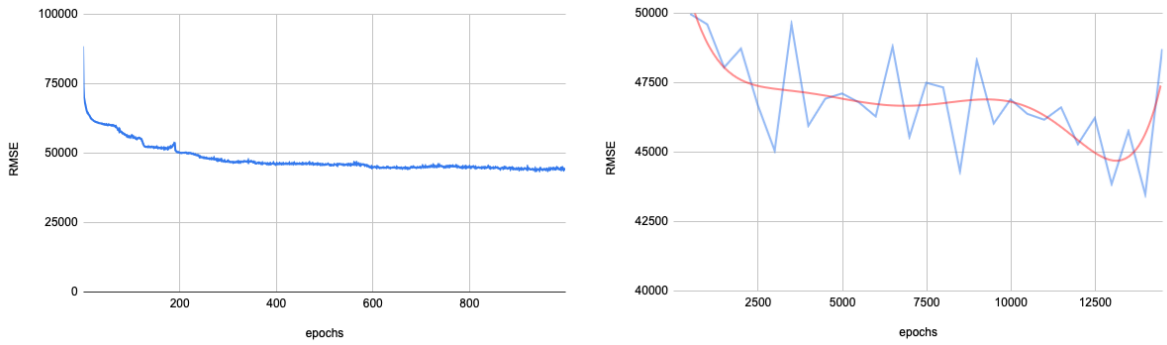
(a) Performance against batch size

Figure 3a shows that the batch size has a significant impact on the performance of the model, with lower batch sizes leading to an increase in accuracy during cross-validated training. This is to be expected as a smaller batch size will lead to more granular gradient descent. However, a small batch size will impact performance negatively by reacting to noise in the dataset. This can be seen in Figure 3a as the rate of increase in performance decreases as the batch size decreases.

4 Final Evaluation

Batch Size	Hidden layers sizes	Learning rate	Epoch	RMSE
500	[64, 32]	0.1	10000	50429

Table 2: Optimal Hyperparameters



(a) RMSE decreasing during training

(b) Overfitting with too many epochs

After the hyperparameter grid search has been performed, the optimal hyperparameters are returned, these are shown in Table 2. Constructing a *Regressor* using the optimal hyperparameters, the model is trained using all available training data (90% of the total data). Figure 4a shows the MSE per epoch as the model is trained. In order to prevent overtraining, the model was evaluated against the test dataset after being trained with a certain amount of epochs. Figure 4b shows that the model starts to overtrain after around 12000 epochs of training. So our choice of 10000 epochs of training is a good choice to maximise accuracy while ensuring that the model we output is not overtrained on the dataset.

In order to properly evaluate the model, the withheld test dataset which the model has never encountered is used to find the average Root Mean Squared Error across the test samples. Our final error on the test data is 50429 which is close to the RMSE on the training data of 44172. This indicates that the model is well-trained and generalised for the entire dataset.

References

[1] Zuowei Shen, Haizhao Yang, and Shijun Zhang. "Neural network approximation: Three hidden layers are enough." In: *Neural Networks* 141 (2021), pp. 160–173. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2021.04.011. URL: https://www.sciencedirect.com/science/article/pii/S0893608021001465.