Appendix

# Glossary

* Bagging – using sampling with replacement from the data to create artificial training data (bootstrap samples)
* Bayesian Optimisation – a function used to run through an array of different hyperparameters for a machine learning model
* Decision Tree Regression – splits the data into distinct regions with the region’s mean being the response to a prediction [2] (paraphrased)
* Hyperparameter – a parameter of the machine learning model (so named as machine learning models are used to find the parameters of a model)
* Lasso Regression
* LFGS
* Linear Regression: an algorithm which attempts to fit an equation of the form Y = wX + c to data so that its error amount to the true value is minimised as far as possible
* Normalised: the data is shifted so that its mean is zero and rescaled to move its standard deviation to 1. I am excluding the zero/one categorical columns from this, as it doesn’t make sense to include them.
* Random Forest Regression – a collection of decision tree regressors (hence the forest) that behaves as a single model as the average prediction is returned from the collection of decision tree results. Each individual tree is trained using “bagging and random features” [1]. A random subset of features is given to each tree.
* Ridge Regression

# Intermediate results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Type** | **Hyper-Parameters** | **RMSE** | **MAE** | **Accuracy (1-NMSE)** |
| Linear Regression | Default (columns: model, year, mileage, fuel type, MPG, engine size) | 4,782.14 | 3,602.55 | -0.04 |
| Linear Regression | Default (columns: model, year, mileage, fuel type, MPG, engine size), Normalised | 2,427.54 | 1,717.89 | 0.73 |
| Linear Regression | Lambda', 0.0002972, 'Learner', 'leastsquares', 'Regularization', 'ridge', 'Solver', 'bfgs' (columns: model, year, mileage, fuel type, MPG, engine size), Normalised | 1,828.18 | 1,346.70 | 0.85 |
| Linear Regression | Lambda', 0.0002972, 'Learner', 'leastsquares', 'Regularization', 'ridge', 'Solver', 'bfgs' (columns: model, year, mileage, fuel type, MPG, engine size), Normalised, with validation data | 1,837.15 | 1,350.96 | 0.85 |
| Linear Regression | Lambda', 0.000010015, 'Learner', 'leastsquares', 'Regularization', 'ridge', 'Solver', 'bfgs' (columns: model, year, mileage, fuel type, MPG, engine size), Normalised, validation | 1,859.48 | 1,358.33 | 0.84 |
| Linear Regression | Lambda', 0.000010015, 'Learner', 'leastsquares', 'Regularization', 'ridge', 'Solver', 'bfgs' (columns: model, year, mileage, fuel type, MPG, engine size), Normalised | 1,826.26 | 1,347.21 | 0.85 |
| Random Forest | Default (columns: model, year, mileage, fuel type, MPG, engine size) | 1,279.66 | 889.33 | 0.93 |
| Random Forest (technically boosted forest) | MinLeafSize, 31, Method, LSBoost, NumLearningCycles, 193, LearnRate 0.38119. (columns: model, year, mileage, fuel type, MPG, engine size). | 1,185.69 | 837.56 | 0.94 |
| Random Forest | MinLeafSize, 31, Method, Bag, NumLearningCycles, 193, (columns: model, year, mileage, fuel type, MPG, engine size). | 1,398.75 | 968.24 | 0.91 |
| Random Forest | MinLeafSize, 1, Method, Bag, NumLearningCycles, 499, (columns: model, year, mileage, fuel type, MPG, engine size). | 1,175.02 | 838.02 | 0.94 |
| Random Forest | MinLeafSize, 1, Method, Bag, NumLearningCycles, 450, (columns: model, year, mileage, fuel type, MPG, engine size). | 1,175.03 | 838.63 | 0.94 |
| Random Forest | MinLeafSize, 1, Method, Bag, NumLearningCycles, 50, (columns: model, year, mileage, fuel type, MPG, engine size). | 1,178.42 | 839.35 | 0.94 |
| Random Forest | MinLeafSize, 1, Method, Bag, NumLearningCycles, 500, (columns: model, year, mileage, fuel type, MPG, engine size). | 1,174.68 | 838.28 | 0.94 |

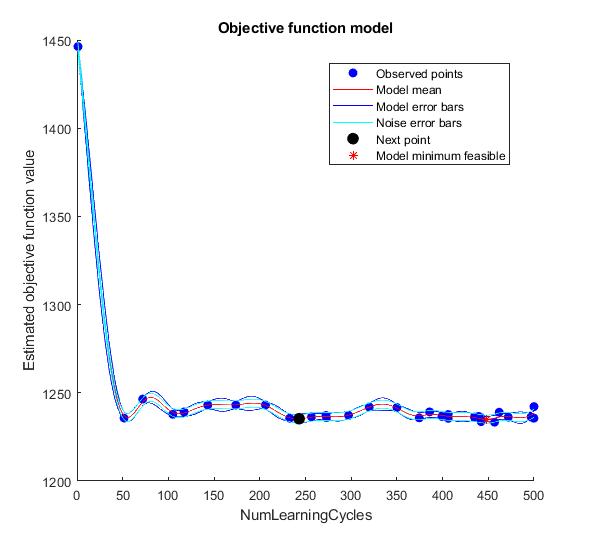


Figure RF Num Learners V RMSE

From the above graph you can see that there is not much difference between using 50 trees and 500. I have kept it at 499 as the results against the test set were better (if we cared about computational efficiency we might cap at around 50 trees).

# Implementation Details

For the linear regression I am excluding tax and transmission type columns and normalising the data. By normalising the data, the residuals then jumped into a proper normal distribution (before they were rather significantly more on the negative side). There are still many residuals to the left, giving the histogram a very slight right skew (in picture NormalisedLRResidualPlot.jpg in case the final plot is different).

Using the script OptimizeLinearRegression I got the general hyperparameters for the Linear Regression model. I then tried using Bayesian optimisation to find the lambda, but that figure performed worse than that found by the auto-tuning (though not by much).

For random forest I used the function fitrensemble (which fits any combination of learners) to train a collection of decision trees (I used a template tree so that I could adjust the minimum leaf size). I then used the script OptimiseRandomForest to find the best hyperparameters for the model.

To fairly analyse both models I used the function analyseRegression to get the residual plots, MAE, RMSE and other graphs.

To get the summary statistics I used the function GetSummaryStats, which I took a lot of inspiration from Pandas’ describe function. Where I built up a matrix, transformed it into a table and printed it out.

By using the script main I could run both models one after the other and get MATLAB to run their statistics and print them to the command window. I used the rng function to set the random number seed so that the models would output the same numbers each time (this has the risk of the models being trapped in local minimums rather than finding the best parameters. Running with a seed of 52 rather than my usual 42 I get RMSE 1826.14 for Linear Regression and 1174.61 RMSE for Random Forest, so not too significantly different).

My first choice was whether to use a Live Script (notebook style) or an ordinary script. I chose the ordinary script so that I could easily rerun the entire script top to bottom. I had it wipe the command window, clear all of the saved variables and close all of the graph output windows; this was so that previous runs wouldn’t affect the next runs.

My next choice was to run the data transformation each time the script runs, this would be inadvisable with a larger dataset where it would be best to save these as a new dataset. This did allow me to experiment with different transformations without destroying the original data.

As I had quite a lot of data, I didn’t run n-fold cross validation on it (this would involve having n separate mini-models trained and tested on subsets of the data).

Throughout the model I had the script delete temporary variables, this was to save memory space.

The script is divided up into various subsections so that they can be run independently, as you can in live scripts.

# References

1. Schapire Robert (2001) *Random Forests*, *Random Forests*. Available at: <https://link.springer.com/content/pdf/10.1023/A:1010933404324.pdf> (Accessed: 1 December 2021).
2. James, Gareth, Witten, Daniela, Hastie, Trevor, and Tibshirani, Robert. *An Introduction to Statistical Learning with Applications in R*. 2nd ed. Vol. 1. 1 vols. Chapter 8: Springer Science+Business Media LLC, 2021. <https://ebookcentral.proquest.com/lib/city/reader.action?docID=6686746>.