School of Electronics and Computer Science

COMP6208: Advanced Machine Learning

Group Project Machine Learning Report

Adam Putland - 29516552 - ap1e17

Luke McClure - 29573904 - lam3g17

Aran McConnell - 29131855 - ajm2g17

Abstract

In this report we explore applying different machine models to the Used Car Sales dataset from Kaggle [1] with data preparation steps already applied.

1 Introduction

This report will discuss the second phase of this Machine Learning Project. This phase carries directly on from the work discussed in our Data Exploration report, and uses the Dataset prepared in phase one (based on the Used Cars Dataset [1]) as a starting point. This phase of the project is focused on exploring a range of Machine Learning regression techniques with the aim of producing a model that can predict used car prices as accurately as possible.

Before any investigation was carried out, our dataset was randomly sampled into train and test sets in an 80:20 split. These two sets were then saved to disk as separate CSVs, and for investigation and tuning only the train set was accessed. The test set was then accessed to produce a final accuracy figure for our best model.

2 Short-List Promising Methods

To begin we first explored training many simple and quick models on the dataset with no tuning. This will allow us to short list the promising methods for further tuning or ensembling. It will also show us the strengths and weaknesses of different models. A range of models were chosen to solve this regression problem, ranging from simple linear regression, to multi-layer perceptrons and some ensembling methods. For each model, we performed K-cross validation and computed its mean and standard deviation of the mean squared error (MSE) on the K folds. We chose $k = 10$ as this provides lot's of training data and still maintains large validation sets. This value also gives a good balance between computation time and introduced bias [11]. Table 1 shows each model and these respective measurements.

Model	Mean	Standard deviation
LinearRegression [3]	2.52×10^{17}	6.33×10^{17}
Lasso [4]	0.011420	$2.72\text{x}10^{-8}$
Ridge [5]	0.0047802	$1.25\text{x}10^{-8}$
MLP [6]	0.002921	$7.38\text{x}10^{-9}$
Decision Tree [7]	0.003789	$1.07\text{x}10^{-8}$
Random Forests [8]	0.002214	$6.14\text{x}10^{-9}$
Adaboost [9]	0.011164	$1.29\text{x}10^{-5}$
XGBoost [10]	0.002785	$6.61\text{x}10^{-9}$

Table 1: Table showing mean and standard deviation of MSE obtained from models using N-fold cross-validation

From this table it is clear which models worked well and which didn't. Of particular note is the first simple linear regression model which had an extremely high MSE and standard deviation. We believe this is because it doesn't contain any regularization so the weights it is learning are climbing extremely high. This then results in extreme overfitting and an inability to generalise at all. Clearly, adding regularization to this, shown in the Lasso and Ridge models, stopped this overfitting and explosion of weight values.

Next, we plotted the distribution of MSE of the most promising models (all apart from base linear regression). This can be seen in Figure 1.

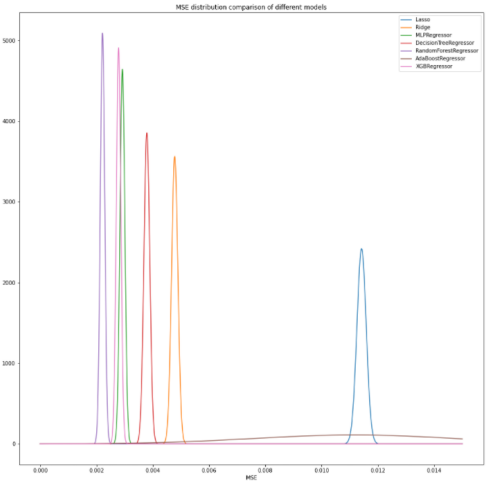

Figure 1: Graph showing MSE distribution of top models over k-fold cross validation

Figure 1 provides a great visualisation of the performance of each model. At a first glance, this graph shows that the majority of the models have quite a small variance, with the exception of the AdaBoost regressor. This extremely high variance translates as the model overfitting to a very high degree. Adaboost also has a higher

mean MSE compared to other techniques. The next outlier in this graph is then Lasso Regression. Whilst the standard deviation is acceptable, it's mean is much higher than the average, indicating that it significantly under performs alternative models here. The final cluster of regressors have relatively similar mean and variances, with RandomForestRegressor, MLPRegressor, and XGBoost all standing out as having the highest performance.

We also examined the most influential features for each model. Most models provided an in-built feature importance attribute which we could examine apart from the linear regression models. For these, we used the features with the highest associated weights/coefficients. Table 2 shows the top 3 most important features for all models apart from the MLP. This is because it is notoriously hard to interpret the feature importance's in neural networks.

Model	Top 3 most important features
LinearRegression	'title_status_parts only', 'title_status_salvage', 'title_status_rebuilt'
Lasso	'region_lake charles', 'state_wy', 'region_kansas city'
Ridge	'year', 'manufacturer_ferrari', 'manufacturer_morgan'
Decision Tree	'odometer', 'year', 'drive_fwd'
Random Forests	'odometer', 'year', 'drive_fwd'
Adaboost	'odometer', 'year', 'condition'
XGBoost	'drive_fwd', 'fuel_gas', 'year'

Table 2: Table showing the top 3 most important features of trained models

For models that are shown to not have good performance through the last graph, such as LinearRegression and Lasso, this table shows that they pick out interesting features as their most important. For LinearRegression, the "title_status" feature is shown to be most important. Whilst this feature definitely has an effect on price, it will give extreme swings as for example "parts only" will result in a huge drop in value. This feature can therefore be very helpful for some extremes in price, but generally for the dataset it will not be that useful. This could explain the huge overfitting and errors given by LinearRegression in Table 1. Lasso on the other hand takes state and region features to be the most important. Through intuition it is clear that these features cannot be the biggest indicators of a car's price, thus indicating overfitting. The Ridge regressor results are the most intuitive so far, with year being picked out as most important, which is likely true. But then manufacturer being Ferrari or Morgan is picked out next. These two manufacturers are expensive car brands so this is a good indicator of extreme values in the dataset, but these features will be useless for the majority of data as these two car brands are rare and represent extremes.

The rest of the models had much more intuitive important features, with odometer, year, and drive type being most prevalent. These features are clearly a very important indicator of vehicle price, and this relationship with price should hold true for almost all vehicles. These "sensible" features give a good indicator that these models will perform, and generalise well.

Using the results of Figure 1, we will be taking forward Random Forests, XGBoost and MLP to finetune in order to explore the performance of each model fully.

3 Fine-tuning the system

Once the top three default models were identified, these models were explored in more detail to fine tune them and gain maximum performance for the problem. This fine tuning consisted of experimenting with each model independently, varying the model hyperparameters, and measuring the resulting performance.
In order to try a good spread of hyperparameters, but to keep computation time manageable, a few key hyperparameters were chosen for each model, and these were varied using a random search method with 60 searches and 10-fold cross validation. RandomizedCVSearch from scikit-learn [2] was used to carry this out.

3.1 MLP

There are a number of hyperparameters to consider when training an MLP network, within this tuning we focused on:

- Hidden layer number - the number of hidden layers within the network
- Hidden layer width - The width of each layer within the hidden layers
- Solver - the solver used to perform gradient descent
- Learning rate schedule - controls how the learning rate changes throughout training
- Initial learning rate - the starting learning rate

- Maximum iterations - the maximum training iterations
- Early stopping - if the trainer should attempt to detect overfitting and halt training.

The values chosen for numeric hyperparameters typically explored +/- 1-2 stops of the default value, depending on the nature of the hyperparameter these stops were orders of magnitude (for initial learning rate) or 100's (for layer width and maximum iterations). Categorical hyperparameters such as solver, learning rate schedule, and early stopping explored every category available.

The best MLP regressor found had one hidden layer of 500 units, trained for 300 iterations using ADAM with early stopping, all other hyperparameters were default. This resulted in a final validation MSE of 0.0029326.

3.2 Random Forests

When tuning the Random Forest Regressor [8], three parameters were chosen to vary in the random search. These parameters were:

- Max Depth - the maximum depth of the tree allowed.
- N Estimators - the number of trees in the ensemble.
- Min Leaf Samples - the minimum number of samples required at a leaf node.

The Max Depth was varied from "None" to 50 in increments of 10. For this hyperparameter the default value is "None", meaning that there is no limit on how deep the tree will run, so it will run until all leaves are completely pure. Having this value as "None" may cause the model to overfit, so it was varied to the effect different depth limits would have on performance. The N Estimators hyperparameter was varied between 100 and 800, where 100 is the default value. Increasing this value will greatly increase computation time, but will help smooth out variance in the model due to it ensembling with more trees. Min Leaf Samples was given values of 1, 5, 10, and then through to 60 in intervals of 10. The higher this is, the more forgiving each leaf of the tree is, requiring less purity to continue. This should have the effect of smoothing the model.

The validation MSE of the the best hyperparameter set was 0.0237871. This best hyperparameter set ended up being the default RandomForestRegressor parameters; "None" max depth, 100 estimators, and a min leaf samples of 1.

3.3 XGBoost

When tuning XGBoost, three hyperparameters were chosen:

- Max Depth - the maximum depth of the tree allowed.
- N Estimators - the number of trees in the ensemble.
- Learning rate - step size shrinkage

The max depth parameter had a range from 10 to 50 in increments of 2. The N estimators parameter had a range from 100 to 800 in steps of 50 and finally, the learning rate was varied in order of magnitudes from 1 to 0.000001. These ranges gave a good trade-off between parameter range and performance (as testing over 800 estimators took over 10 minutes per model).

This best model had a validation MSE of 0.0207998. The parameters for this model were a max depth of 15, N Estimators equal to 800 and a learning rate of 0.1.

3.4 Ensembling

Now that each individual model had been fine tuned for the training data, an ensemble of them could be explored to see if there is was performance gain. The validation MSE was measured over 10-fold cross validated sets for all combinations of fine-tuned models. This gave the results in table 3.

Model	Mean	Standard deviation
MLP and XGBoost	0.000584	$1.03\text{x}10^{-5}$
MLP and Random Forests	0.000503	$9.70\text{x}10^{-6}$
XGBoost and Random Forests	0.000011	$9.09\text{x}10^{-7}$
MLP and Random Forests and XGBoost	0.000262	$4.7\text{x}10^{-6}$

Table 3: Table showing mean and standard deviation of MSE obtained from ensembled fine-tuned models

All combinations of models performed much better than them on their own. Interestingly, XGBoost and Random Forests ensembled together performed better than all 3 together, with a lower mean MSE and standard deviation.

3.5 Final Model

The final mode chosen was XGBoost ensembled with Random Forests since this gave the best mean validation MSE and lowest variance. Each model was then retrained on the entire training data set with the parameters found through fine-tuning. With this final model defined, the test set can be used (as mentioned in section 1) to estimate its generalization error. This resulted in an MSE of 0.00229. This is slightly larger than the validation error in training (see Table 3), which is as expected and indicates that this model slightly overfits the training set.

4 Results and reflection

The final mode obtained had a mean squared error of 0.00229 on our test set. This model consists of an ensembling of XGBoost and Random Forests (both with fine-tuned parameters). These two methods are often very successful so it is no surprise that they ended up in the final model. As shown in Table 2, these methods also pick up on different features. With XGBoost picking up on 'drive_fwd', 'fuel_gas' and 'year' and Random Forests additionally focusing on 'odometer'. These features intuitively are the most important for pricing a car and the ensembling of both methods ensure they are all taken into account for the price estimation. It is hard to decide what is the most important feature across these two models, but the existence of both 'year' and 'drive_fwd' in both of their top 3 feature importance's indicate that those 2 features are the most pivotal in the dataset.

Compute was a big limitation throughout this stage of the project, largely because of the size of the dataset. If this was not a limitation, or if there was more time available, a full grid search for each model would be carried out, therefore testing a greater number and range of hyperparameters. A larger number of hyperparameters could then be varied and not such a select few. Due to memory limitations and the large dataset, some of the investigation was carried out on subsets of the dataset as the full dataset could not be loaded into memory. This may have led to some results varying from what their true values if the entire dataset was used. Whilst any final numerical results used the full dataset, some decisions on models to pursue were made using smaller subsets.

Despite the necessary shortcomings within both the fine-tuning and pre-processing stages of development, it is clear that the ensemble model created is a strong predictor of car value that is able to generalise well to this problem.

References

[1] "Used Cars Dataset", Kaggle.com, 2021. [Online]. Available: https://www.kaggle.com/austinreese/craigslist-carstrucks-data. [Accessed: 16- Apr- 2021].

[2] Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

[3] "sklearn.linear_model.LinearRegression — scikit-learn 0.24.2 documentation", Scikit-learn.org, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html. [Accessed: 07- May- 2021].

[4] "sklearn.linear_model.Lasso — scikit-learn 0.24.2 documentation", Scikit-learn.org, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html. [Accessed: 07- May- 2021].

[5] "sklearn.linear_model.Ridge — scikit-learn 0.24.2 documentation", Scikit-learn.org, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html. [Accessed: 07- May- 2021].

[6] "sklearn.neural_network.MLPRegressor — scikit-learn 0.24.2 documentation", Scikit-learn.org, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html. [Accessed: 07- May- 2021].

[7] "sklearn.tree.DecisionTreeRegressor — scikit-learn 0.24.2 documentation", Scikit-learn.org, 2021. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html. [Accessed: 07- May- 2021].

[8] "sklearn.ensemble.RandomForestRegressor — scikit-learn 0.24.2 documentation", Scikit-learn.org, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html [Accessed: 07- May- 2021].

[9] "sklearn.ensemble.AdaBoostRegressor — scikit-learn 0.24.2 documentation", Scikit-learn.org, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html [Accessed: 07- May- 2021].

[10] "XGBoost Documentation — xgboost 1.5.0-SNAPSHOT documentation", Xgboost.readthedocs.io, 2021. [Online]. Available: https://xgboost.readthedocs.io/en/latest/index.html. [Accessed: 07- May- 2021].

[11] Brownlee. "How to Configure k-Fold Cross-Validation", https://machinelearningmastery.com/how-to-configure-k-fold-cross-validation/.