Automobile Price Prediction using Regression

Louisa Giapoulis, Saroop Samra, Yuling Zhang

Department of Management Information Systems, San Diego State university

MIS/BA 749: Business Analytics

Dr. Aaron Elkins

December 14, 2021

Executive Summary

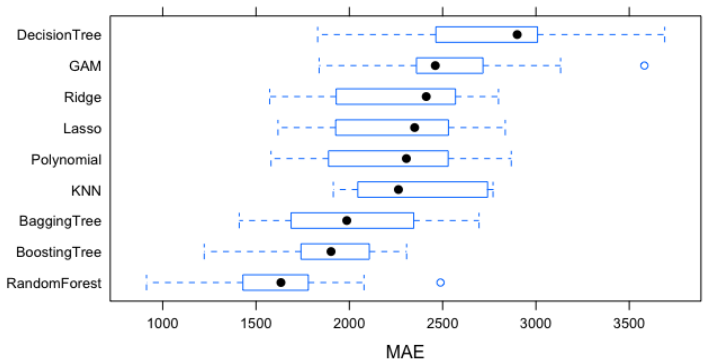
Our analysis is related to predicting the price of an automobile. We used the dataset from the UCI database "1985 Auto Imports Database"; this dataset has multivariate characteristics and includes 26 variables that consist of both categorical and numerical attributes (Schlimmer, 1987). Our goal is to determine which regression algorithm best predicts the price attribute. The prediction of price is important for customers as the price is one of the most important facts when buying a car. Additionally, for companies to make the best possible judgment on setting prices they need to know what variables are related to customers willing to pay for higher prices. Thus, manufacturers can analyze which feature is worth the price and which ones are not. This is especially the case with rising competition among car producers and the new technological changes this knowledge is key. By discovering which type of regression classifies price the best, we can provide manufacturers with helpful information for the best type of prediction.

To accomplish this we first explored the data and imputed any missing values as well as encoding any categorical variables as quantitative ones to use the attributes for our analysis with Dummy Coding (otherwise known as One Hot Encoding). We continued to explore our data by performing EDA where we created multiple visualizations such as boxplots, scatter plots, and histograms. After the data exploration was done, we did a test train split and calculated several regression models including Linear Regression, Polynomial regression, Generalized Additive Model, Decision Tree, Bagging Tree, Boosting Tree, Random Forest, Lasso regression, and Ridge regression. For each model, we performed k-fold cross-validation with 10 folds and calculated the error metrics MSE (mean squared error), RMSE (root mean squared error), MAE (mean average error), and R-Squared on the test data. Through a comparison of errors in the bar

charts and box plots, we determined which models were best, which we found were Random Forest and Boosting Tree, seen for example in Figure 1 below that shows the MAE.

Figure 1

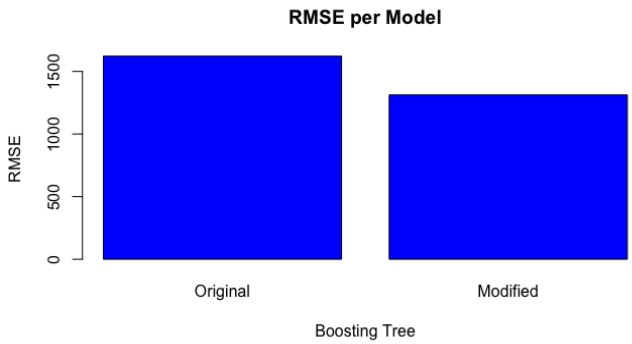
Comparison of MAE



We decided to tune these in order to get the best models possible. We did this through a grid search with different possibilities for each attribute in the model. Our outcome helped us understand how to set the parameters of the models for the best possible model in which we got Boosting Tree to perform the best; see for example in Figure 2 below which shows the RMSE.

Figure 2

Boosting Tree RMSE



Discovery and Data Preparation

We searched through UCI's database and found an automobile dataset that we thought would be interesting to look at and understand the relationship between the different properties. The N size (or sample size) is 205 and there are 26 attributes. Our goal with this dataset is to determine which type of regression analysis is best in predicting *price*. The business case for this is useful for both the manufacturer and the buyer. This is because it can allow the manufacturer as well as the buyer to understand better which parts contribute to a higher-priced car. To prepare this data, we used Python to clean the data. First, we observed the dataset to see how it is currently organized.

Currently, the name of the columns in the header is numerical values rather than the appropriate column names. To fix this problem, we cleaned up the table by adding the appropriate column names to each column. We also converted whichever columns should be classified as numeric to the numeric type. The next step is to perform some EDA, exploratory data analysis, on our data to get a better understanding of what the data looks like and the relationship between the attributes. We first described the data as a preliminary step to comprehending our data seen in Table 1 below.

Table 1

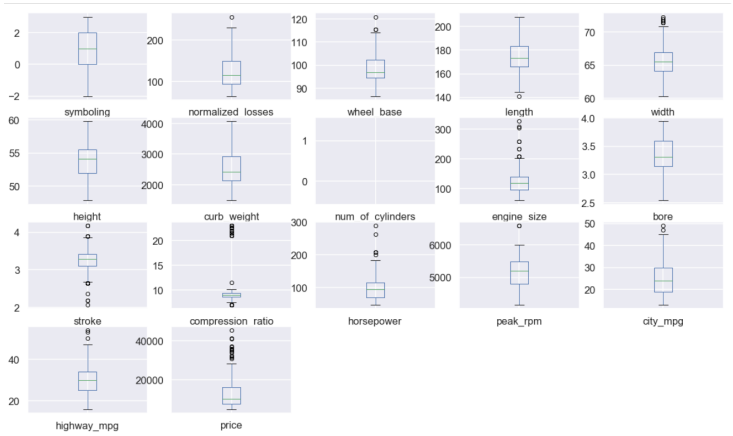
Dataset Statistic Description

	symboling	normalized_losses	wheel_base	length	width	height	curb_weight	num_of_cylinders	engine_s
count	205.000000	164.000000	205.000000	205.000000	205.000000	205.000000	205.000000	0.0	205.000
mean	0.834146	122.000000	98.756585	174.049268	65.907805	53.724878	2555.565854	NaN	126.907
std	1.245307	35.442168	6.021776	12.337289	2.145204	2.443522	520.680204	NaN	41.642
min	-2.000000	65.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	NaN	61.000
25%	0.000000	94.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	NaN	97.000
50%	1.000000	115.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	NaN	120.000
75%	2.000000	150.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	NaN	141.000
max	3.000000	256.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	NaN	326.000

We then did some univariate analysis by creating box plots for the attributes shown in Figure 3 below. Based on the graph, the most interesting charts are *engine_size*, *compression_ratio*, *horsepower*, and *price* which all have several cases above the upper whisker.

Figure 3

Box Plots for each attribute



We also created some histograms. Figure 4 is plotting the *stroke* attribute, which shows that there is a major peak at around 3.4 strokes with a frequency of close to 35. Figure 5 describes *horsepower* and shows that there is a massive peak at around 60 horsepower with a frequency over 40. Figure 6 shows the *normalized_losses* which has a peak at around 95 with a frequency over 20 and a subsequent peak at 150 with a frequency over 15.

Figure 4

Stroke Histogram

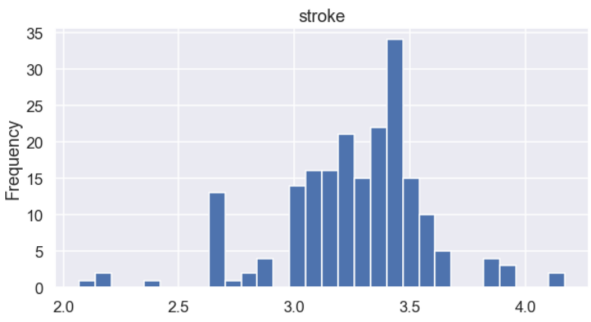


Figure 5

Horsepower Histogram

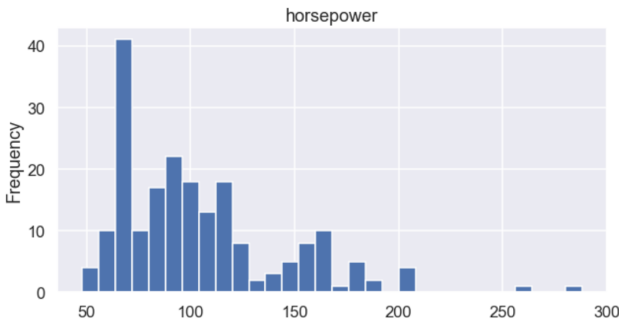
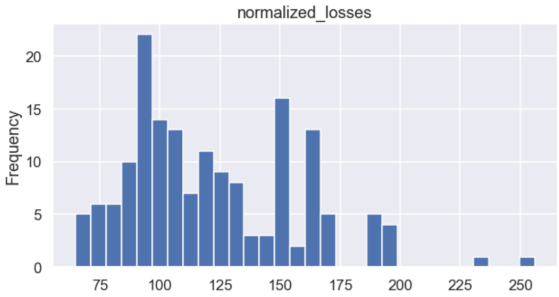


Figure 6

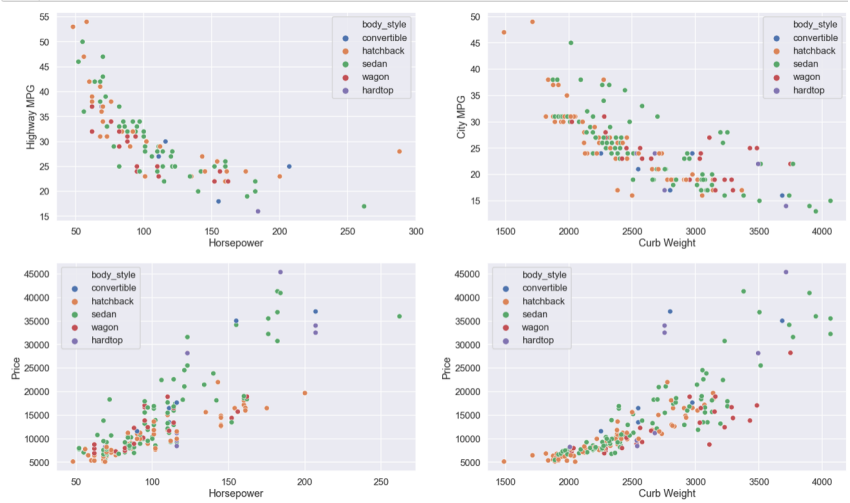
Normalized Losses Histogram



We created several scatterplots for bivariate analysis. As seen in Figure 7 below, there are 4 subplots each showing a different relationship. For all charts, the color is marked by the *body_style*. The top-left graph shows *horsepower* vs *highway_mpg*, which appears to be a positive decreasing quadratic relationship. The top right graph shows the *curb_weight* vs *city_mpg*, which is nearly a negative linear relationship. The bottom left graph shows *horsepower* vs *price* which is generally a linear positive increasing relationship. The bottom right graph shows *curb_weight* vs *price* and shows a quadratic positive increasing relationship.

Figure 7

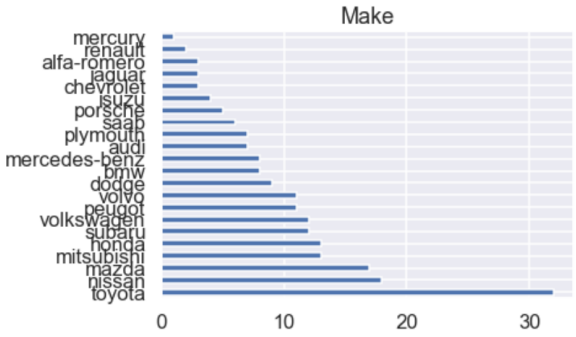
Relationship Between Variables



We created a bar plot to get a closer look at the *make*, shown in Figure 8 below. We see that the most popular make in this dataset is Toyota, followed by Nissan and Mazda.

Figure 8

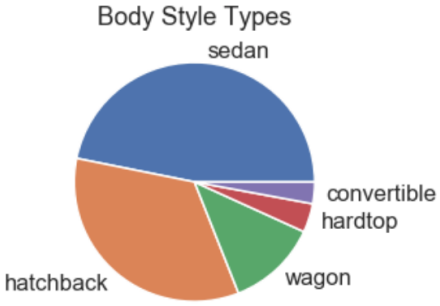
Popularity of Makes



We also wanted to comprehend the *body_style* types better and thus created a pie chart as seen below in Figure 9. We can see that Sedan is the most popular type of *body_style* in our data set followed by hatchbacks.

Figure 9

Body Style Types

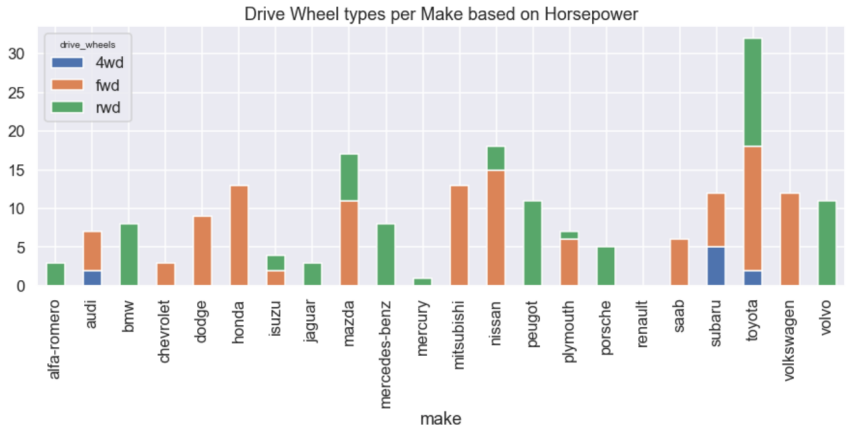


The next chart is a stacked bar plot in which we looked at the *drive_wheels* types per *make* based on *horsepower*. For Toyota, which has the most cases, we see that it has the most *fwd* (front wheel drive) but also a close number of *rwd* (rear wheel drive cars). There are not many cases of *4wd* (4 wheel drive cars) in general. Based on our chart we see that several makes are explicitly

front wheel drive while other makes are only rear wheel drive, but some makes like Toyota produce all 3 different types of *drive_wheels*.

Figure 10

Drive Wheel Types Per Make Based On Horsepower



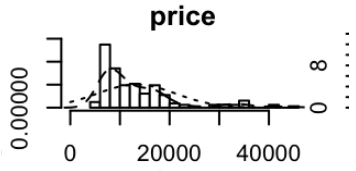
This completes our EDA for the dataset, which leads to the next step of dummy encoding (also called one hot encoding). We have to perform dummy encoding for all the categorical variables in order to use them in our regression analysis. The columns that required this were the *make*, *fuel_type*, *aspiration*, *num_of_doors*, *body style*, *engine_type*, *fuel_system*. Then we dropped these columns from the table and added the new dummy encoded version of the columns. The last step is to save this new table as a new csv, which will be the table we use in the model creation and prediction steps. We imported the table into R and reviewed the data structure with str, names, head, and summary functions.

Taking a deeper look into the data in R, we noted that the column *num_of_cylinders* is empty, therefore, we removed this column. We also have supplemental code to make sure the columns are of the numeric type. Additionally, we imputed all the columns with the mean of the column. After the imputation step, we created a correlation matrix to understand which variables are highly correlated. We don't want to use highly correlated variables as that can cause problems

with our modeling. Based on the 'findCorrelation' function, we found attributes *curb_weight*, *length*, *horsepower*, *highway_mpg*, *rwd*, *diesel*, *gas*, *idi*, *two*, *std*, *peugot*, *ohcf*, *rear*, *X1bbl*, and *X4bb* are all highly correlated. Price was also an attribute marked as highly correlated, but this makes sense as this is what we plan to predict based on only the other attributes in the dataset. We removed these highly correlated variables (apart from *price*) to allow for good modeling. Another step we took to ensure the best data is removing low variance, close to 0, variables from our data. These variables have the potential to cause problems in our models which are X4wd, *front*, *alfa*, *romero*, *audi*, *bmw*, *chevrolet*, *dodge, isuzu, jaguar, mercedes.benz, mercury, plymouth, porsche, renault, saab, X., convertible, hardtop, dohcv, rotor, mfi, spdi,* and *spfi*. The majority of these columns are our dummy encoded variables which makes sense as these do not have much data and therefore much variance in them. We also found that most of our columns do have a skewness factor, as we see the "price" below is right-skewed in Figure 11. However, because most of the variables are skewed we will leave our data as is, though this might be a potential limitation in our modeling and prediction.

Figure 11

Example of Skewness in Data



Based on our understanding of the dataset, we assume that nonlinear models should offer a more accurate prediction for our analysis. We then hypothesize that GAM, due to its capacity to model complicated relations with a large number of predictors, and Random Forest, with its ability to reach high accuracy with the approach of utilizing multiple Decision Trees, are the best models in predicting the price of an automobile.

Model Planning and Building

The first model we worked on creating was a Linear Regression model. We first fit our Linear Regression model with cross-validation using the 'train' function on all the variables in order to predict price. We looked at which variables are most important, notably *engine_size* is most significant at 100 followed by *hatchback* at 41.28. We reviewed the actual and residual observations against the predictions seen in Figure 12 and 13 below. These plots show a linear trend between the actual and predict as well as generally flat residuals which is what we want.

Figure 12

Actual vs. Predict

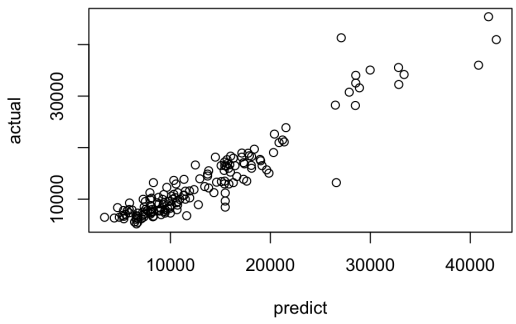
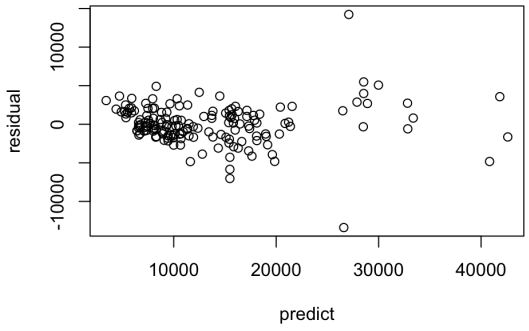


Figure 13

Residual vs. Predict



The errors we got from this model on the test data were 12012566 for MSE, 12012566 For RMSE, 2594.578 for MAE, and 0.7941914 for R-Squared. These are generally not the best scores so we will continue to work with different regression models and compare at the end. The second model we tested as a possible method is Polynomial Regression. We fit the model and split our data in training and testing. We determined a MSE training error of 62025475, a MAE training error of 5754.197, a RMSE training error of 7875.625 and training R-Squared of 0.005377212. The error results for the test data are 53179862 for MSE, 7292.452 for RMSE, 5503.062 for MAE and the R-Squared is 0.08888123. We then used k-fold cross-validation and defined the training control as cross-validation and set the value of K equal to 10 as we will use

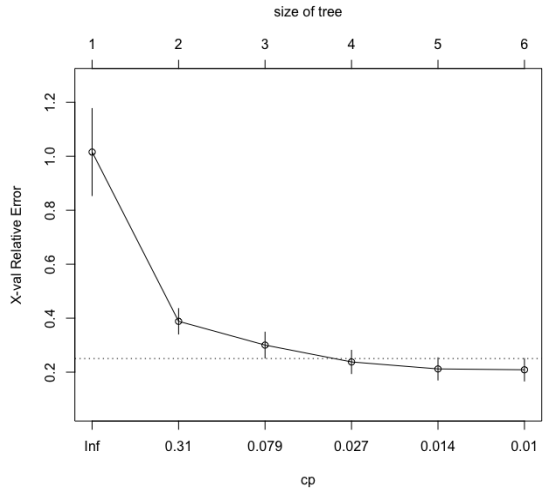
for all our models. Next we trained the model by assigning the *price* column as target variable and the other variables as independent ones. As the last step, we then printed the model performance metrics along with other details. From this we got our most important error metrics with the test data where MSE is 7163806, RMSE is 2676.529, MAE is 2000.116, R-Squared is 0.8772641.

In addition, we used GAM, the generalized additive model, as a potential model. We used the split training and testing data for the following error metrics. We got the MSE training error which is 7233025 and the RMSE training error of 2689.428 which is pretty high. The result of the MAE training error is 1811.955 and the R-Squared of the training error is 0.8840133. For the test data, we got the MSE of the test data is 7163806, RMSE is 2676.529, MAE is 2000.116 and R-Squared is 0.8772641. Finally, we printed the model performance metrics along with other details; using the test data we got 7163806 for MSE, 2676.529 for RMSE, 2000.116 for MAE, and 0.8772641 for R-Squared.

We also used Decision Tree as another machine learning algorithm, which we first created by hand followed by using the cross-validation method. We first created the tree and summarized it. We then plot this to see how the Decision Tree looks. However, in order to have less variance as it likely overfits, we find the cross-validated error per tree size. We plot the cross-validated error shown in Figure 14 below. We see that 0.01 has the lowest error amount.

Figure 14

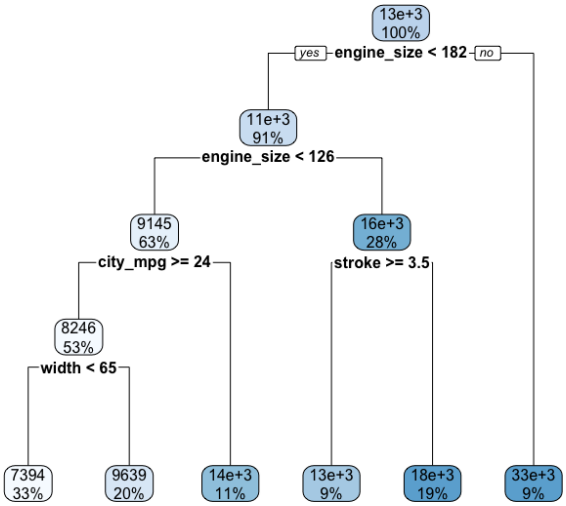
CV error



Next, we prune the tree, which is a reduction of complexity by removing parts that are not important in the Decision Trees which will improve the accuracy. We plot the final tree after pruning shown below in Figure 15 below. Based on the chart, there is 91% of the data where the *engine_size* is less than 182. The other 9% of the data is where *engine_size* is greater than or equal to 182. Within the 91%, 63% of the data is where *engine_size* is less than 126. The other 28% is where the *engine_size* is greater than or equal to 126. Of the 63%, 53% percent is where *city_mpg* is greater than or equal to 24; within this, there is 33% of the data where *width* is less than 65, and 20% of the data is where *width* is greater than or equal to 65. The other 11% is where *city_mpg* is less than 24. The other 28% is split by *stroke* where there is 9% of the data based on when *stroke* is greater than or equal to 3.5 and 19% of the data is where *stroke* is less than 3.5.

Figure 15

Final Tree



We gathered error metrics based on our train and test data on the Decision Tree that we created manually. We then did cross-validation and got our test metrics which are 14473628 for MSE, 3804.422 for RMSE, 3026.395 for MAE, and 0.7520265 for R-Squared.

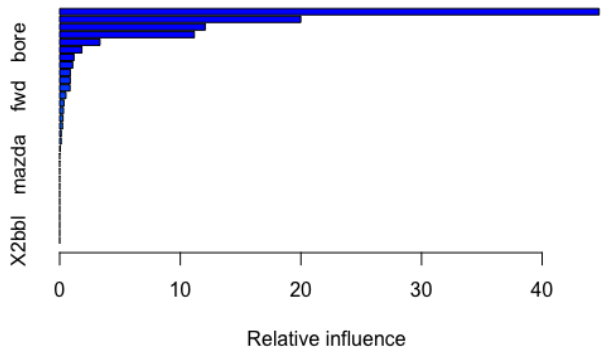
The next algorithm we perform is Bagging, which is bootstrap aggregation that helps reduce noise and variance done in parallel. By using the 'randomForest' library, we were able to set up the Random Forest function with m as all the variables in the dataset, which is 61 as there are 61 predictors. Finally, we determine the MSE training error which is 1693240, the RMSE is 1301.246, MAE is 821.2544, and R-Squared is 0.9728477. Using the test dataset we found that the MSE is 6440750, RMSE is 2537.863, MAE is 1874.326, R-Squared is 0.8896521. We then did our k-fold cross-validation and got the same error metrics on the test data. We decided to do another Bagging algorithm with a subset of the predictors; we chose our m by calculating the square root of 61 which rounded equals 7. After calculating the errors based on the test data from our cross-validation, we saw this is a worse error at 8189110 for MSE, 2861.662 for RMSE,

2040.215 for MAE and 0.8596978 for R-Squared. This means that having a subset of the data did not help for the Bagging algorithm and instead having all the variables is better.

Our other algorithm was Boosting which trains weaker learners sequentially (which means it is a slower algorithm). We used the 'gbm' library to help us create the model with the default parameters for tree depth, tree size, and slowness lambda. We plotted the variable importance as seen below in Figure 16 where we see that the attribute *bore* has the most significance with the highest at over 40 relative influence. What follows *bore* is *fwd*, *mazda*, and *X2bbl* in order of most to least important. These attributes following *bore* do not have nearly as much relative influence as bore.

Figure 16

Variable Importance



Finally, we calculated the error metrics on our model that we created by hand. After performing cross-validation on the test data we got the MSE as 2629598, RMSE as 1621.604, MAE as 1252.203, and R-Squared as 0.9549477.

Another model we built based on Decision Trees is Random Forests. Compared with bagged trees, the Random Forest model uses a small tweak that further decorrelated the trees, which greatly reduces the variance when averaging the trees. We still use the library "randomForest" to build this model. We choose the number of predictors considered at each split to be the square

root of the number of total predictors, which, with our automobile dataset, returns 7. We choose the default setting of the number of trees, which is 500. From the output of the model prediction, we could see that the mean squared residuals (MSR) is 7650343, while around 87.73% of the variance could be explained by the model. We then calculate the MSE test error, which is 8189110 for Random Forest. We then performed 10-fold cross validation on the model with the test dataset, and got the MSE as 1811835, RMSE as 1346.044, MAE as 950.2813, and R-Squared as 0.9689582.

We also perform Lasso and Ridge Regression that utilize shrinkage methods. Instead of fitting a linear model that contains a subset of the predictors, this type of model utilizes a technique that constrains the coefficient estimates, which shrinks them towards zero, and thus we could fit them containing all the predictors. This shrinkage technique could also reduce the variance.

Both Lasso and Ridge Regression contain a tuning parameter lambda, and selecting the appropriate lambda is crucial to the performance of both models.

For Ridge Regression, we use cross-validation to find the best lambda that minimizes the test MSE error, and generate Figure 17, a trace plot to show how the coefficient estimates changed as the lambda increases, and Figure 18, a plot of test MSE by the lambda value. We then performed 10-fold cross validation on the model with the test dataset, and got the MSE as 10685828, RMSE as 3268.919, MAE as 2468.963, and R-Squared as 0.8169221

Figure 17

Ridge Trace Plot

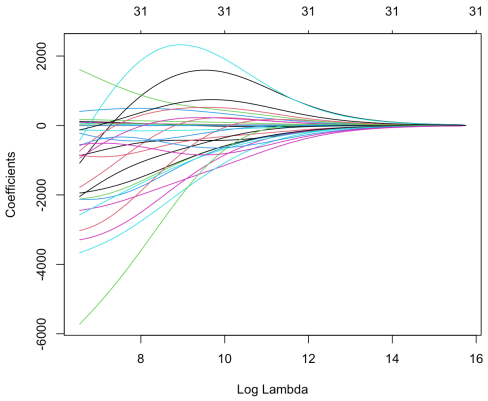
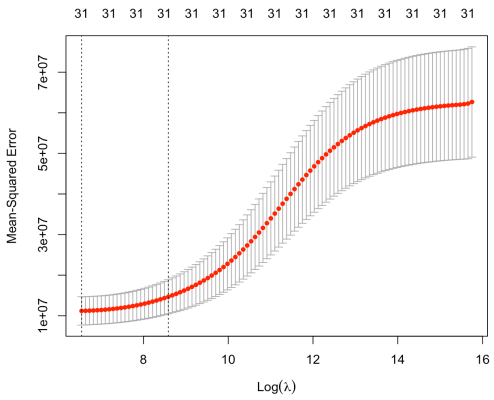


Figure 18

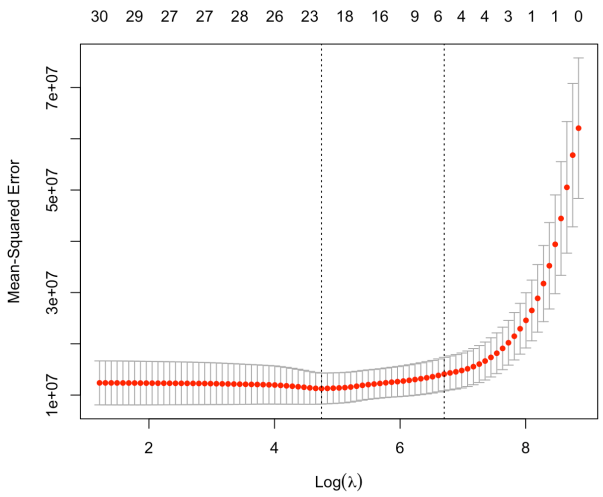
MSE vs. Lambda (Ridge)



Being slightly different from Ridge Regression, the Lasso would force some of the coefficient estimates to be exactly equal to zero, when the tuning parameter is large enough. We also generate Figure 19, a plot of the MSE by the lambda value.We then performed 10-fold cross validation on the model with the test dataset, and got the MSE as 7543288, RMSE as 2746.505, MAE as 2061.111, and R-Squared as 0.8707625.

Figure 19

MSE vs. Lambda (Lasso)



Results and Performance

We plotted the errors we got from the cross-validation and compared them using bar plots to get a better understanding of the distribution across the models. Our four bar plots in Figure 20, 21, 22, and 23 show the different errors which from left to right are our Linear Regression model, Polynomial Regression, GAM model, Decision Tree, Bagging Tree, Bagging Tree 2, Random Forest, Lasso Regression, and finally Ridge Regression. We see that Boosting Tree and Random Forest perform the best out of all the models as they have the lowest MSE, RMSE, MAE and highest R-Squared values. The GAM model is the next best after Random Forest and Boosting.

Figure 20

MSE Comparison

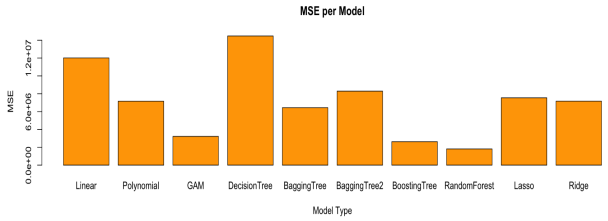


Figure 21

RMSE Comparison

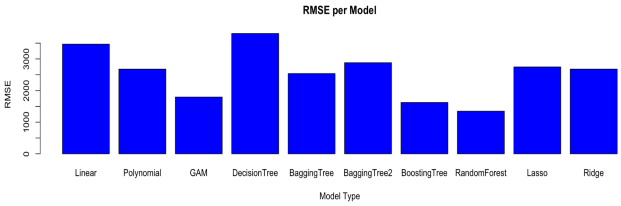


Figure 22

MAE Comparison

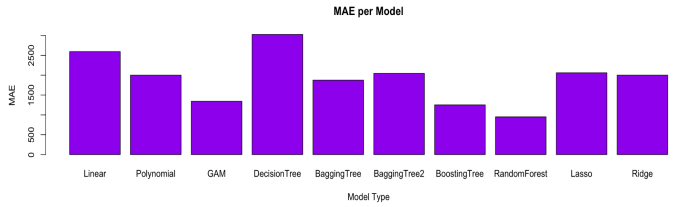
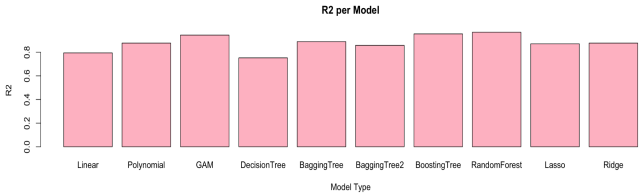


Figure 23

R^2 Comparison

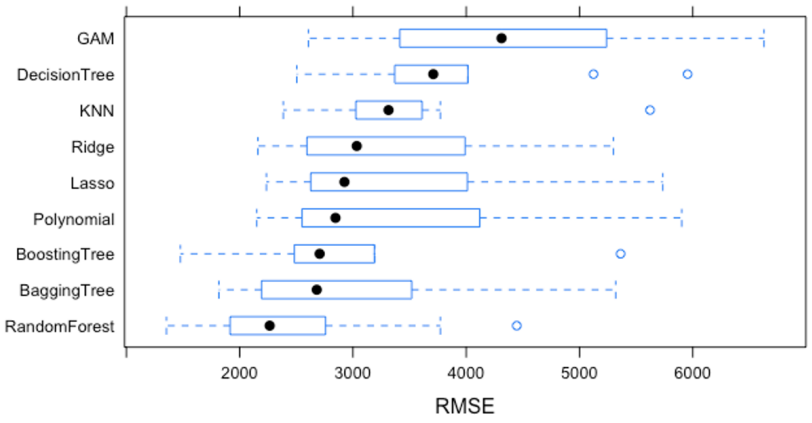


We created box plots in order to understand the mean of the error metrics for each model, its variance, and how they compare against each other. Additionally this will help us confirm our findings from our bar plots above. The first box plot seen in Figure 24 below shows the RMSE. As we can see, Random Forest has the lowest RMSE with an error slightly over 2000. Bagging

Tree and Boosting Tree are the next best models with similar RMSE values. However, Bagging Tree has much longer length of the whiskers which represent the variance. Boosting Tree on the other hand has better whisker lengths and therefore less variance. The other models such as Polynomial, Lasso, Ridge have very high variance. The GAM model performs the worst with a RMSE over 4000.

Figure 24

RMSE of Models



The next box plot shown in Figure 25 below compares the MAE. We see again that Random Forest and Boosting Tree perform best with the lowest mean average error. Both models have similar variance, however, it appears Random Forest has slightly more variance, while Boosting Tree has less variance. These two models remain the best as the other models have higher MAE with similar amounts of variance. We also looked at the R-Squared calculation shown in Figure 26. The higher R-Squared the better as this means more of the variance can be explained and that the model fits the data better. Similar to what we saw in our RMSE graph, Random Forest, Bagging Tree, and Boosting Tree perform the best. However, Bagging Tree has very high

variance in comparison to Boosting Tree. Therefore, Random Forest and Boosting Tree perform best with R-Squared values around and above 0.9.

Figure 25

MAE of Models

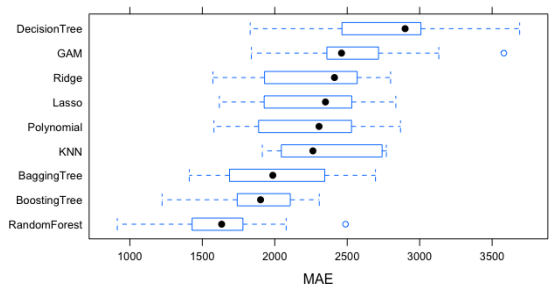
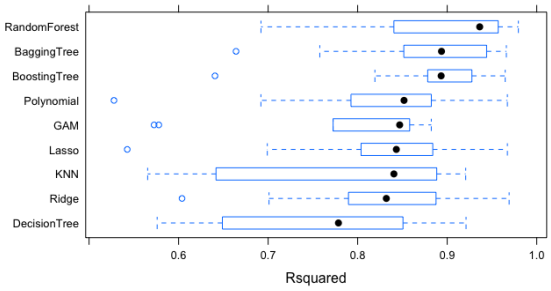


Figure 26

R-Squared of Models



The results of the analysis are largely consistent with our hypothesis, with Boosting and Random Forest providing the best results, and GAM following closely behind. Being one of the algorithms that utilize numbers of Decision Trees, Bagging doesn't decorrelate the trees and thus reduces the variance as the Random Forest does, also it doesn't grow trees sequentially so that the next tree could use the information of the previously grown tree as the Boosting algorithm does, hence it fell short after the other two. We also contemplate that GAM doesn't perform as well because although it allows for flexible nonlinearities in some variables, it retains the additive structure of linear models, so it doesn't fit as appropriately.

We proceed in modifying both the Boosting Tree and Random Forest models in an attempt to get even better results and conclude with our best fitting model for our data. We did our fine tuning by creating a grid search on the different properties for the model. By adding this grid search into our train function for the model, we can find out which variables allow for the best model and then compare the error metrics. First we decided to tune our Random Forest model by searching for the best ‘mtry’, which represents the number of variables for splitting at each node. We searched from 1 to 20 as possibilities for mtry and found that we get the best results when mtry

equals 7. The RMSE for a Random Forest model with mtry as 7 is 1358.87, while the MAE is 986.54, and R-Squared is 0.965315. We tested different amounts of ‘mtry’ including up to 50, however, we found the best results when we stuck to 1 to 20. We compare this metric to our outcome from the cross-validation model we did earlier as we see in Figures 27, 28, and 29 below. The cross-validation model on test data had errors of 1346.044 for the RMSE, 950.2813 for MAE, and 0.9689582 for R-Squared. This means the tuning of our Random Forest model got very similar results, however the cross validated model where ‘mtry’ is 16 is best, as we see in the bar plots below.

Figure 27

RMSE of Original and Modified RF

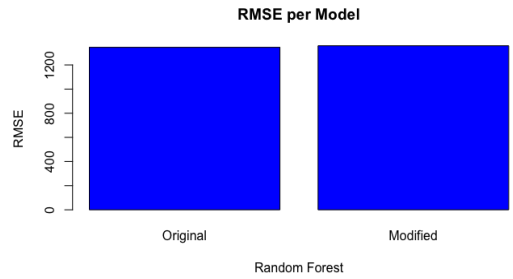


Figure 28

MAE of Original and Modified RF

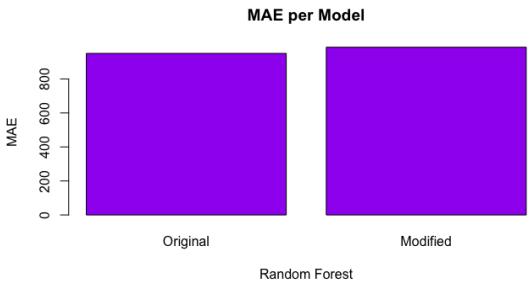
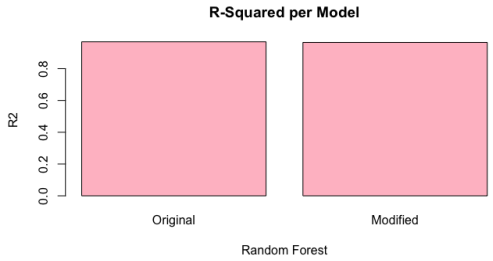


Figure 29

R-Squared of Original and Modified RF



We also looked at the Boosting Tree model in an attempt to get the best model possible. In our grid search we looked for the from 5 to 100 every 5th number for the number of possible trees

trees, 1 to 10 for the interaction depth, .0001, .001, .01, .5 for shrinkage, and 1, 10, or 20 for number of minimum samples in the nodes. The best model is where n.trees is 25, interaction.depth is 14, shrinkage is 0.1, n.minobsinnode is 1. The errors for this model have an RMSE of 1457.061, MAE of 1129.753, and R-Squared of 0.9836467. By looking back at our errors from our cross validated we had RMSE as 1312.061, MAE as 941.753, and R-Squared as 0.9549477. The comparison between our original model and the newly modified one is shown in Figures 30, 31 and 32 below. This indicates that our fine tuning of our Boosting Tree model helped get a better Boosting Tree model.

Figure 30

RMSE of Original and Modified Boosting Model

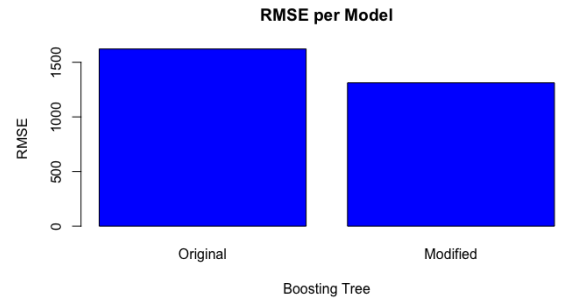


Figure 31

MAE of Original and Modified Boosting Model

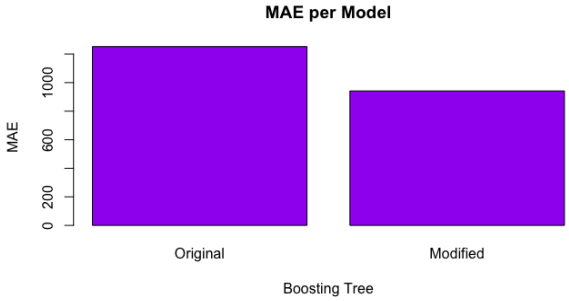
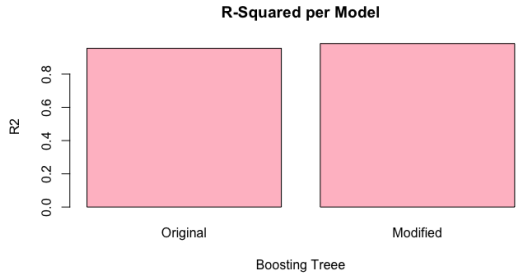


Figure 32

R-Squared of Original and Modified Boosting Model



Finally we look at our results between the cross validated Random Forest model and our modified Boosting model, in which we see that the modified Boosting tree model performs the best and therefore is our best model.

Discussion and Recommendations

We can conclude that our modified Boosting Tree, where number of trees is 25, interaction depth is 14, shrinkage is 0.1, and number of minimum samples in nodes is 1, performs the best and therefore is the best model for our data to predict the price attribute. In our proposal problem statement we thought understanding which attributes best predict price is the best way of understanding our problem, however, we came to the realization that we can understand more about our data by calculating multiple regression models and determining which of those is best. Boosting is a very intricate model as it has many attributes, therefore we had to run our grid search many times in order to get the best results. It also is a slower model as it calculates the trees sequentially rather than in parallel which means the algorithm takes more time. For next steps based on our results, we recommend for more data to be collected and used, including a variety of data for example from different countries. We suggest that more data cleaning is done and that a similar process to ours is taken to determine the best regression model for predicting price with the new gathered data.

References

Agrawal, R. (2021). *Evaluation Metrics for your regression model!*. https://www.analyticsvidhya.com/blog/2021/05/know-the-best-evaluation-metrics-for-your-regression-model/.

CFI. (2021). *What is the adjusted R-Squared?*. https://corporatefinanceinstitute.com/resources/knowledge/other/adjusted-R-Squared/.

DataTechNotes. (2019). *Regression Model Accuracy (MAE, MSE, RMSE, R-Squared) Check in R*. https://www.datatechnotes.com/2019/02/regression-model-accuracy-mae-mse-rmse.html.

DataCamp. (n.d.). *Impute: Generic Functions and Methods for Imputation.* https://www.rdocumentation.org/packages/Hmisc/versions/4.6-0/topics/impute.

Ensemble Methods - Bagging, Random Forests, Boosting (n.d.). https://quantdev.ssri.psu.edu/sites/qdev/files/09_EnsembleMethods_2017_1127.html.

Iannone, R. (2020). *Great looking tables: gt (v0.2).* https://www.rstudio.com/blog/great-looking-tables-gt-0-2/.

Jenkob, J. (2020). *R-Load Data*. http://tutorials.jenkov.com/r/r-load-data.html.

Nilesh. (2017). *RPubs - MultiVariable Linear Regression with evaluation and visualization.* https://rpubs.com/nileshvarshney/281010.

Saralaya, R. (2019). *Ml case studies 1 - uci auto data (Linear Regression).* https://numoonchld.github.io/2019/05/21/ml-case-study-1-uci-auto-data.html.