Model Analysis

Jamie Walters

12/6/2021

walters.688, hall.2565, cluff.18, diederich.16

Background

This data set is from kaggle.com. It includes data about used cars that were sold on Ebay-Kleinanzeigen. We are trying to use the data about the cars sold to predict the price they sold for. The data set is in German. Here are some German translations for the factor variables in our data set:

German	English
Privat	Private
Gewerblich	Commercial
Angebot	Offer
Gesuch	Request
Andere	Other
Cabrio	Convertible
Kleinwagen	Small car
Kombi	Station wagon
Automatik	Automatic
Manuell	Manual
Benzin	Petrol
Elektro	Electric
Nein	No
Ja	Yes

We performed some data cleaning to the set in the EDA, which we summarize below.

There were quite a few errors in the data that needed cleaned up. First, we got rid of nrOfPictures, because all of the values were 0 and the description said it was an error. Next we converted the character columns to factor variables. Next, we filtered the yearOfRegistration to be between 1900 to 2022. The minimum was 1000

and the maximum was 9999, which seem to be placeholders. We also filtered powerPS to be less than 1000 because everything after 1000 seems to be fake listings because the prices were so low. Price also had placeholders like 99999999, 12345678, and 0. We filtered price to be greater than 0 and less than or equal to 500000. This was all done in the EDA.

We created a new predictor called daysOnline. This is the number of days the listing was online, which we calculated by subtracting the dateCreated from lastSeen.

Due to some of the limitations of some of the modeling packages in R, we needed to simplify the postalCode variable to a more reasonable number of categories. We combined the data with another kaggle dataset (https://www.kaggle.com/ravikanth/germany-postal-codes-gis-city-district-and-state) to find the state and district for each of the observations. This simplified 8000 different postal codes into 17 states and 400 districts.

Finally, we changed kilometer to a factor variable. If the value was 150,000, we indicated that vehicle had more than 150,000 kilometers. We did this because through our exploratory data analysis, we saw that the largest bin for kilometer by far was for 150,000. We believe that when the data was recorded, any car that had more than 150,000 kilometers on it was recorded as 150,000.

In our EDA, we made the following important conclusions about the data:

Boxplot for seller looks odd for ‘gewerblich’, which means a commercial seller. It turns out there’s only 2 vehicles sold through a commercial seller.

Most of the boxplots have a lot of outliers on the high price end. Most of the boxplots look to be right skewed.

The plot of price vs power looks like there’s a clump in the lower left hand corner of the plot. This may indicate some sort of relationship.

The plot of price vs kilometer is less clear, there may be some sort of polynomial relationship, but it is not as obvious.

From these conclusions, we determined that seller, offerType, powerPS, kilometer, daysOnline, and notRepairedDamage could be the most important variables. These are the variables we chose to start with for our model analysis.

Analysis

Ordinary Least Squares Model

Motivation:

We knew from the EDA we wanted to try LASSO to try to shrink some predictors to 0 since we had so many. Since LASSO stems from OLS, it was only logical to try a linear OLS model first. Here, we use stepAIC to perform backwards stepwise selection based on AIC.

Mathematical Description

In OLS, the values of β_j were chosen by minimizing the following expression:

$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2$

Backwards stepwise selection starts with a model with all features in it. It then considers all models with one feature removed, and picks the best one. If the best model with one feature removed is worse than the model with all features in it, we stop and the model with all features is the best model. We do this many times, removing one more feature each time, to pick the best model. “Best” is determined by the AIC.

AIC = $2k - 2ln(\hat{L})$, where k = the number of parameters and $\hat{L}$ = maximum log likelihood

Assumptions

The main assumption to check with OLS is collinearity. In our EDA, we determined that collinearlity was not an issue in our data set.

```
cor(select_if(train, is.numeric), use = "complete.obs")
```

```
##                      price yearOfRegistration   powerPS daysOnline
## price              1.0000000         0.24957868 0.4943665 0.11005854
## yearOfRegistration 0.2495787         1.00000000 0.1520484 0.02013018
## powerPS            0.4943665         0.15204844 1.0000000 0.08098000
## daysOnline         0.1100585         0.02013018 0.0809800 1.00000000
```

Although this only checks the collinearity for continious variables, the box plots we examined in our EDA did not reveal any collienarity between the factor variables either. Furthermore, logically, none of the predictors should have collinearity concerns, just based off their descriptions.

Model Validation

```
my.best.model <- lm(price ~ vehicleType, data = train)
variable.formula <- as.formula("~ vehicleType + yearOfRegistration + gearbox + fuelTy
pe + brand + seller + offerType + powerPS + kilometer + daysOnline + notRepairedDamag
e")
stepAIC(object = my.best.model, scope = variable.formula)

x.test = model.matrix(price ~ vehicleType + powerPS + kilometer + brand +
                        fuelType + yearOfRegistration + gearbox + notRepairedDamage +
                        daysOnline, test)[,-1]
y.test = test$price

my.best.model <- lm(price ~ vehicleType + powerPS + kilometer + brand +
                      fuelType + yearOfRegistration + gearbox + notRepairedDamage +
                      daysOnline, data = train)
```

Results

```
ls.pred = predict(my.best.model, newx = x.test)
ols_mse = mean((ls.pred - y.test)^2)
```

```
ols_mse
```

```
## [1] 118632135
```

The test error for OLS is extremely high at 118,632,135. We believe that shrinkage methods, performed below, will help lower this high error and be a better method for modeling our data.

LASSO

Motivation:

Because we were unsure of the usefulness of some of the predictors, LASSO is a modeling approach we wanted to try. This is because LASSO will get rid of any of the predictors that are not useful in the model.

Mathematical Description:

LASSO modifies the OLS estimation procedure by minimizing the following expression:

$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p}|\beta_j|$

This penalizes large values of β_j and has the effect of shrinking the coefficients to 0 as $\lambda \to \infty$.

Assumptions:

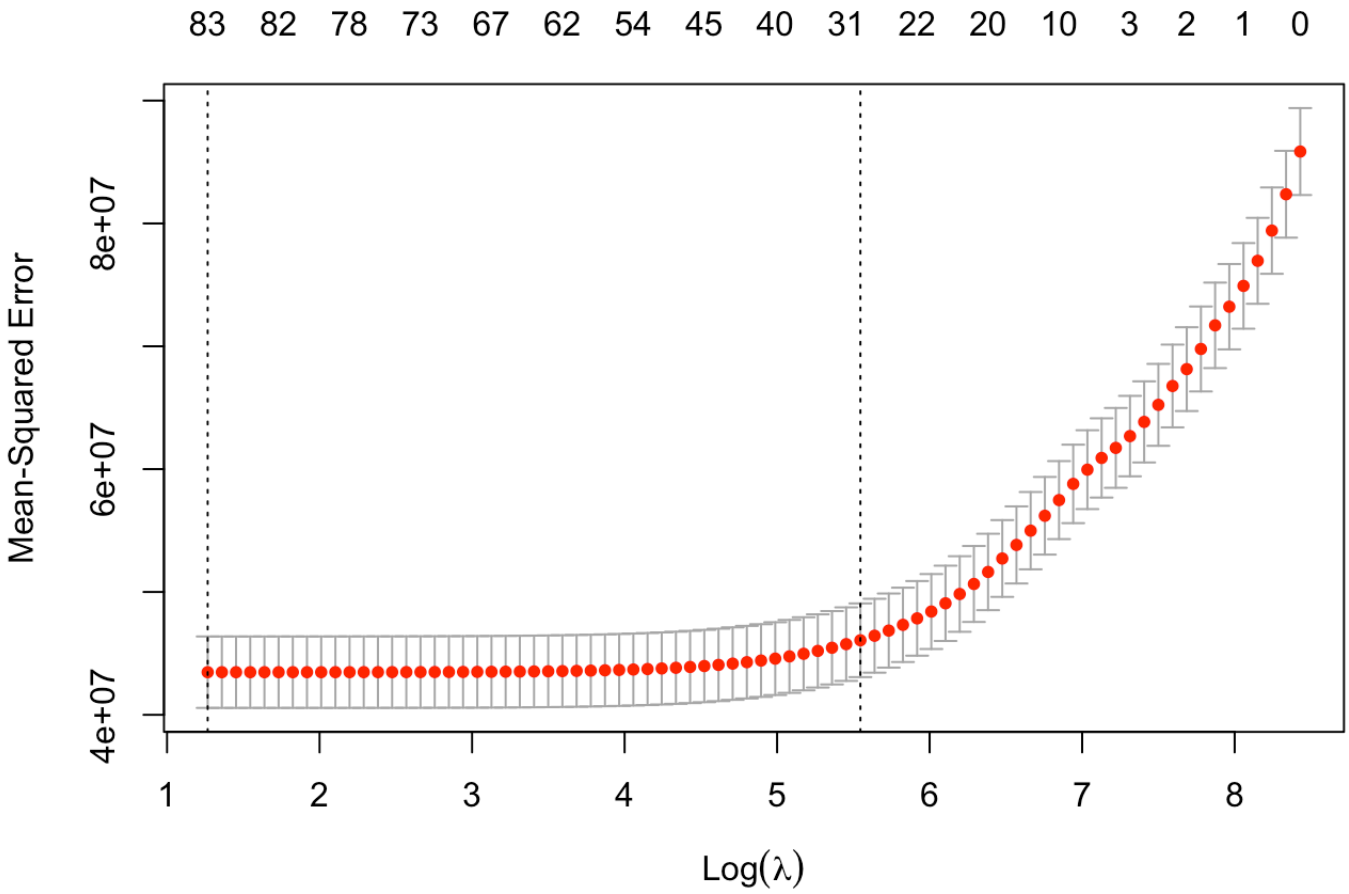
LASSO shrinks OLS estimators, so it has the same assumptions as OLS. Since we already checked that above, we feel comfortable moving on.

Model Validation:

```
## LASSO
x_train = model.matrix(price~vehicleType+yearOfRegistration+gearbox+monthOfRegistrati
on+fuelType+brand+seller+offerType+powerPS+kilometer+daysOnline+notRepairedDamage, tr
ain)
y_train = train$price

x_test = model.matrix(price~vehicleType+yearOfRegistration+gearbox+monthOfRegistratio
n+fuelType+brand+seller+offerType+powerPS+kilometer+daysOnline+notRepairedDamage, tes
t)
y_test = test$price

lasso.cv = cv.glmnet(x_train, y_train, alpha = 1)
plot(lasso.cv)
```




```
lambda.cv = lasso.cv$lambda.min
lambda.cv
```

```
## [1] 3.54823
```

```
fit.lasso = glmnet(x_train, y_train, alpha = 1, lambda = lambda.cv)
```

We used cross validation to select the optimal value of lambda for our model. The value of lambda that minimizes the CV error is 3.54823.

Results

```
pred.lasso = predict(fit.lasso, newx = x_test)
lasso_mse = mean((y_test - pred.lasso)^2)
lasso_mse
```

```
## [1] 36810057
```

```
head(coef(fit.lasso), 10)
```

```
## 10 x 1 sparse Matrix of class "dgCMatrix"
##                                    s0
## (Intercept)           -277544.62968
## (Intercept)                 .
## vehicleTypebus            920.46497
## vehicleTypecabrio        2490.79311
## vehicleTypecoupe         2588.55951
## vehicleTypekleinwagen      57.68991
## vehicleTypekombi         -630.31448
## vehicleTypelimousine     -326.32631
## vehicleTypeNA           -1358.27736
## vehicleTypesuv           2186.55677
```

The resulting test MSE for our model chosen by cross validated LASSO is 36,810,057. Although this may appear high, we do not know if that is the case until we fit other models and compare. The error could be large because the response variable, price, is also large, since it is the price of a car. The chosen model shrunk fuelTypeNA, brandford, brandhonda, and brandrenault to 0, but no others. Furthermore, none of the other coefficients have been shrunk very close to 0 at all. This leads us to believe that LASSO may not be the appropriate method to model our data.

Ridge Regression

Motivation:

It is not true that LASSO always performs better than ridge regression or vice versa. In general, LASSO usually outperforms ridge regression when the coefficients are mostly 0, and ridge regression usually outperforms LASSO when the coefficients are very small, but not exactly 0. Because we are not sure which is true for our data, we decided to perform ridge regression as well, just to cover our bases.

Mathematical Description:

Ridge regression modifies the OLS estimation procedure by minimizing the following expression:

$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p}\beta_j^2$$

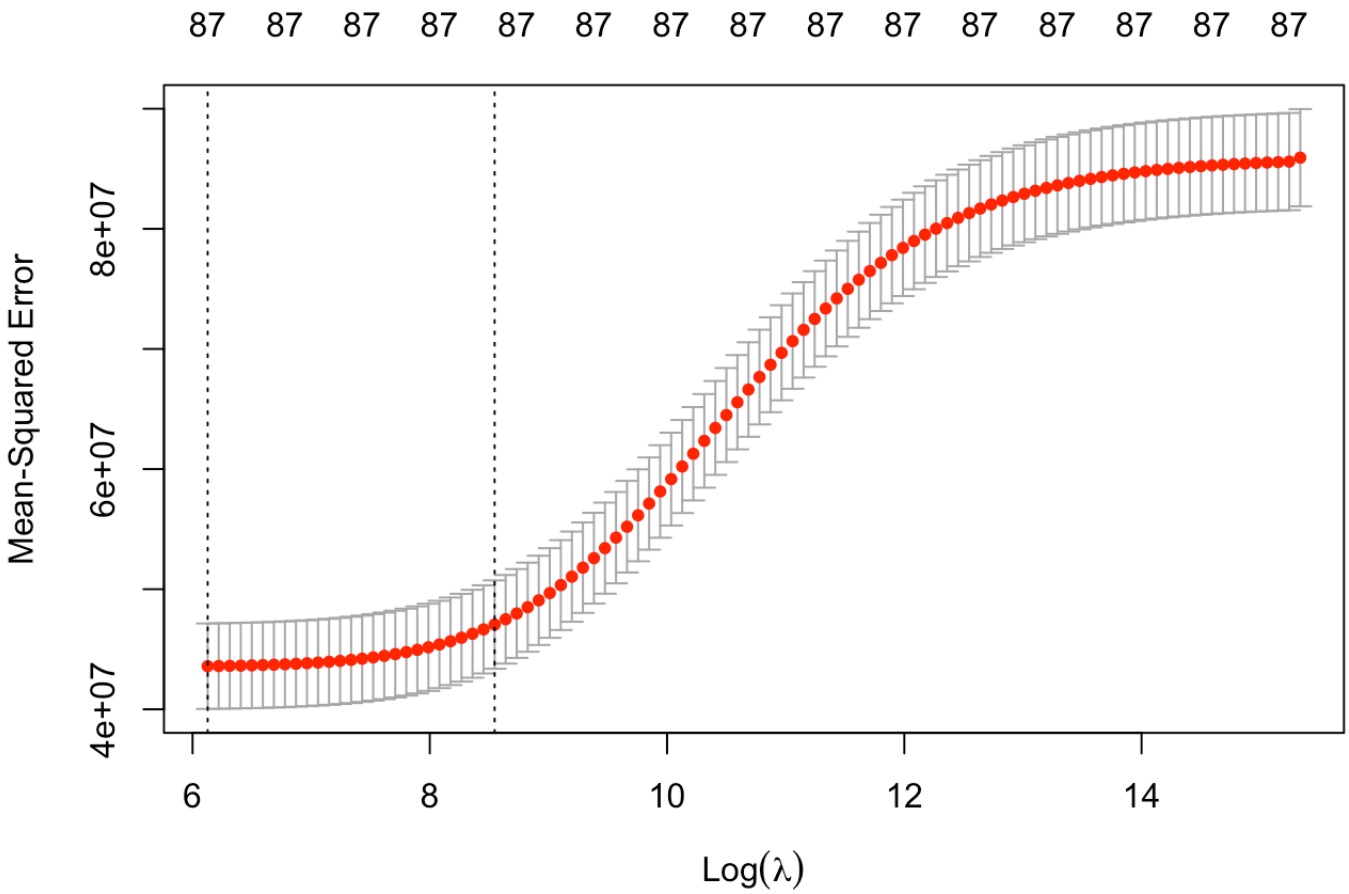
This penalizes large values of β_j and has the effect of shrinking the coefficients to close to 0, but never exactly to 0, as $\lambda \to \infty$.

Assumptions:

Ridge regression shrinks OLS estimators, so it has the same assumptions as OLS. Since we already checked that above, we feel comfortable moving on.

Model Validation:

```
## Ridge
ridge.cv = cv.glmnet(x_train, y_train, alpha = 0)
plot(ridge.cv)
```




```
lambda.cv = ridge.cv$lambda.min
lambda.cv
```

```
## [1] 458.2715
```

```
fit.ridge = glmnet(x_train, y_train, alpha = 0, lambda = lambda.cv)
```

We used cross validation to select the optimal value of lambda for our model. The value of lambda that minimizes the CV error is 458.2715.

Results

```
pred.ridge = predict(fit.ridge, newx = x_test)
ridge_mse = mean((y_test - pred.ridge)^2)
ridge_mse
```

```
## [1] 36767673
```

```
head(coef(fit.ridge), 10)
```

```
## 10 x 1 sparse Matrix of class "dgCMatrix"
##                                s0
## (Intercept)          -282574.0084
## (Intercept)                 .
## vehicleTypebus            732.0702
## vehicleTypecabrio        2342.7044
## vehicleTypecoupe         2459.2692
## vehicleTypekleinwagen    -197.1305
## vehicleTypekombi         -786.3335
## vehicleTypelimousine     -478.9412
## vehicleTypeNA           -1565.0601
## vehicleTypesuv           2101.2347
```

The resulting test MSE for our model chosen by cross validated ridge regression is 36,767,673. This is lower than the test MSE for the model chosen by cross validated LASSO, but not by much. Furthermore, none of the coefficients are shrunk very close to 0 at all. These facts lead us to believe that ridge regression is not a very appropriate way to model our data either.

Random Forest

Motivation/Mathematical Description:

Random forest models have a few benefits compared to other types of trees. Random forests fix the biggest problem with the bagging method: correlation between trees. Random forests is a method aimed at de-correlating the trees before averaging. This is achieved by restricting the number of predictors that can be chosen at each split. In the R-package randomForest this parameter is called mtry. We want the best possible model, so we optimized mtry for lower training error. The biggest downside of a random forest model is the interpretability. A simple regression tree can be visualized easily. Our final tree had 52565 nodes. This would be impossible to visualize. Random forest models also tend to overfit data. Even though we have such a large model, we did not see any evidence of overfitting (see results for details).

Assumptions:

Random forest models require no assumptions. They are nonparametric, so the distribution of the data does not matter. Predictors can also be of any type. In this case we are using both categorical and numerical data.

Model Validation:

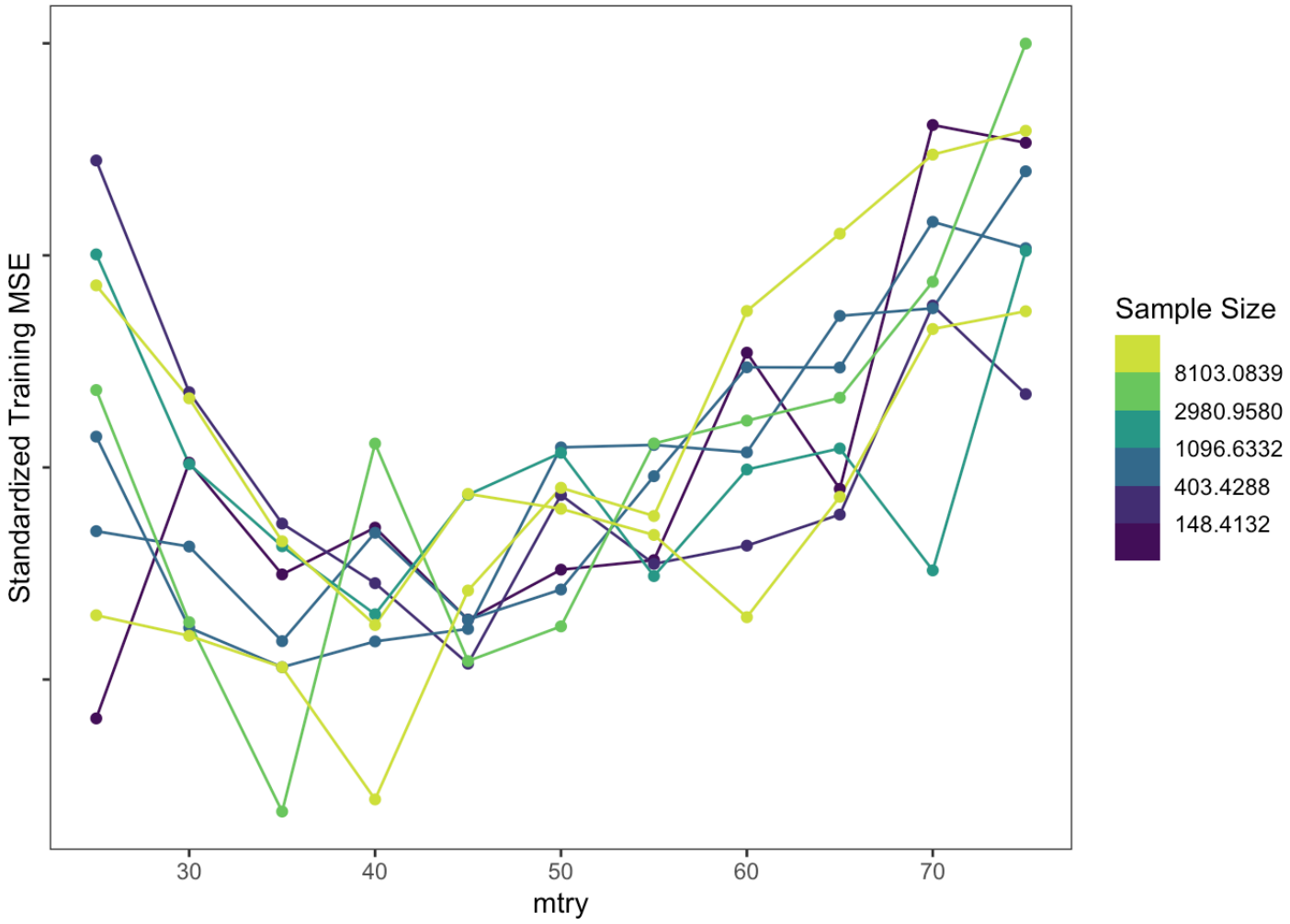
Included	Removed
vehicleType	postalCode
yearOfRegistration	district
gearbox	model
powerPS	seller
kilometer	offerType
monthOfRegistration	lastSeen
fuelType	dateCrawled
brand	dateCreated
notRepairedDamage	
daysOnline	
state	

Although we used seller and offerType in the LASSO and ridge regression models, we had to remove as many predictors as possible in order to speed up the runtime of the model. Removing these two predictors did not change the MSE of the final model by much, so we decided to leave them out of our final model.

One of the limitations of the randomForest R-package is the level limit of 53 for categorical variables. This was most likely for the better because of the increased run-time for variables with a large amount of levels. postalCode, district, and model all had above the allowed levels. postalCode and district were replaced by state.

The three date variables were removed due to their limited range. The data was collected over the span of 2 months. While it might benefit the testing error to leave them in, the error for predicting Ebay listings outside of the two month window would likely be skewed.

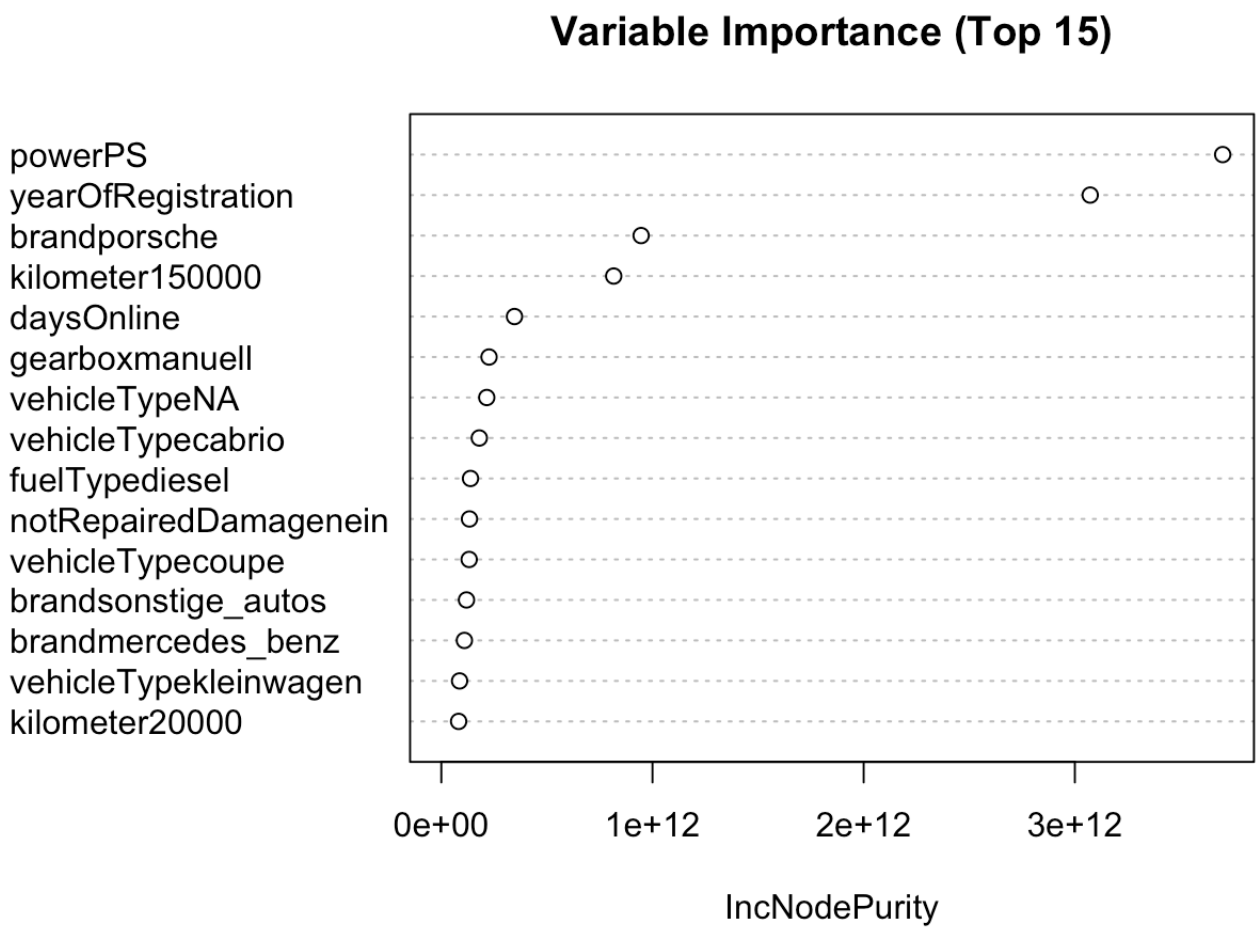
Random forest models are very computationally intensive with large training sets like we have here. The goal is to only run the final modeling process once, so we started by finding the best value for mtry. We modeled many sample sizes over many mtrys. The graph below is comparing the training error achieved by each combination.



We can conclude a few things from this graph. The optimal mtry is around 40 and it is consistent as sample size grows. We can confidently fit the model with mtry = 40 on the full training set.

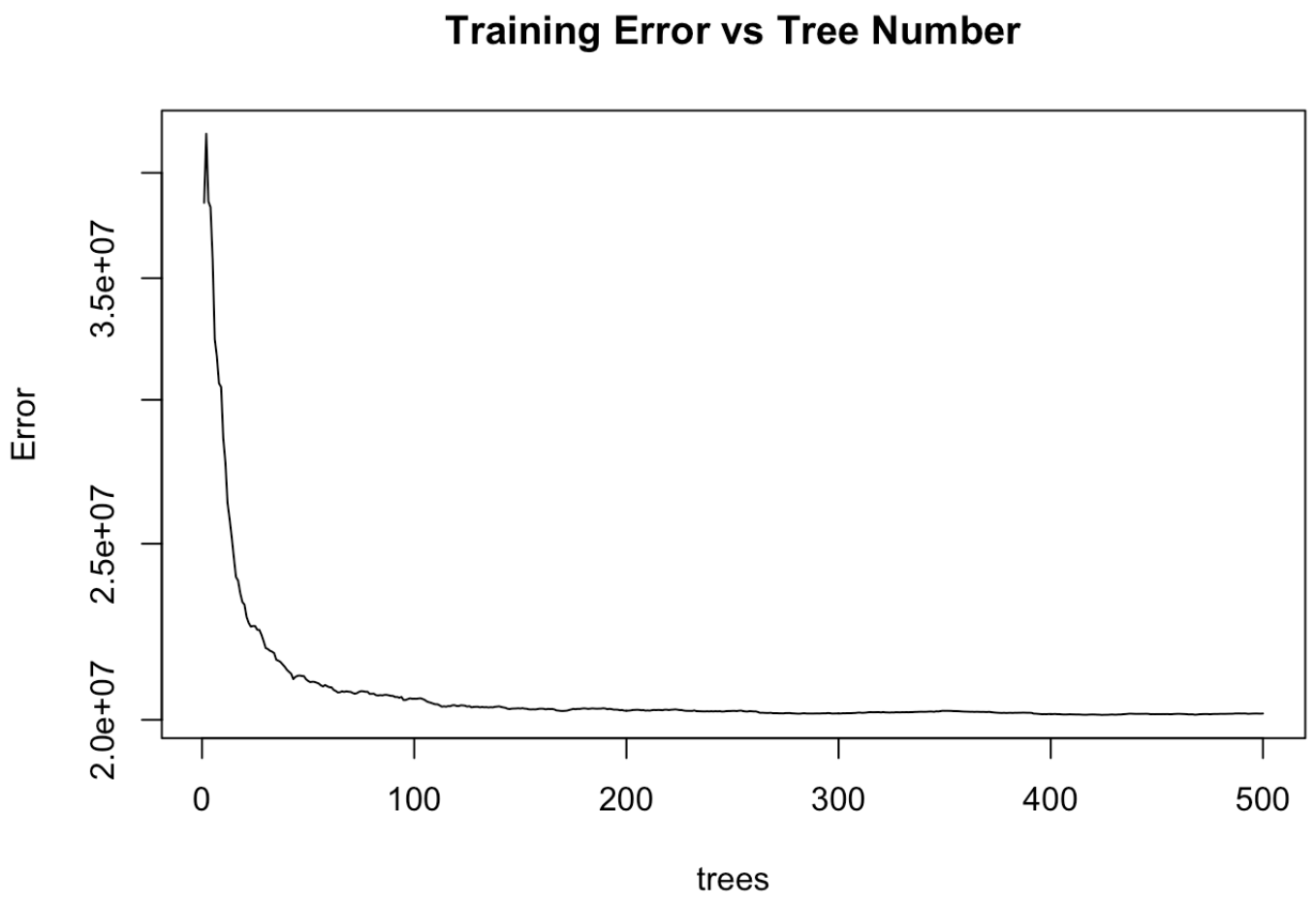
Results

```
varImpPlot(rf.model, main = "Variable Importance (Top 15)", n.var = 15)
```



The top 15 variables are listed above and ranked by importance. Due to the nature of the methods used, categorical variables were split into their own binary predictors. The continuous variables are ranked much higher because of this. Some notable variables are the brand Porsche as well as 150000+ kilometers.

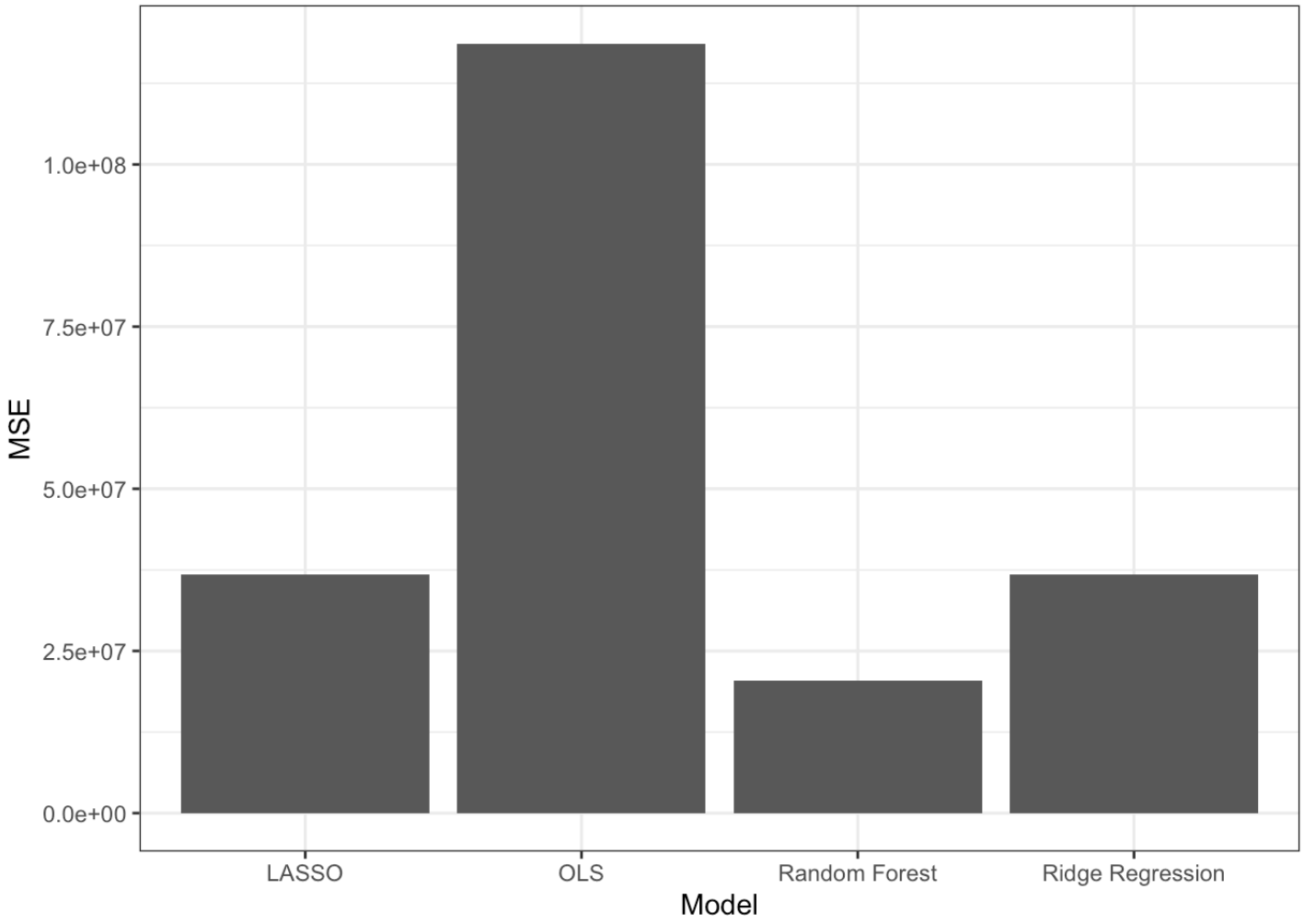
```
plot(rf.model, log="y", main = "Training Error vs Tree Number")
```



Here we can see that the training MSE converges to 20,000,000. The best was at tree 424 with an MSE of 20,124,042. Our final test MSE was 20,410,455. The small difference in MSEs indicates that our model is a very good fit on the data without much overfitting.

Final Results

```
mse = data.frame(Model=c("OLS", "LASSO", "Ridge Regression", "Random Forest"),
                 MSE = c(ols_mse, lasso_mse, ridge_mse, rf.test.mse))
ggplot(data = mse, aes(x = Model, y = MSE)) + geom_bar(stat = "identity")
```



This final test MSE is much lower than the test MSE's of the models created using OLS, LASSO and ridge regression. Because there was so much of an improvement but the MSE was still very high (in the millions), we believe that, indeed, the reason the MSE's are so high is because the response variable, price, is high. We conclude that the random forest model we created is the best fit for our data of all the modeling methods we tried.