module3_part2

Andrew Estes

4/2/2022

```
library(tidyverse)
library(caret)
library(pROC)
```

1

Use the same code that I used in the lab to clean the data frame and get it ready to use. You'll be using the mycars data frame that the lab code defines.

```
data("cars")
brand.names <- c("Buick", "Cadillac", "Chevy", "Pontiac", "Saab", "Saturn")
car.types <- c("coupe", "hatchback", "sedan", "wagon")

mycars <- cars %>%
  mutate(Brand = factor(case_when(
    Buick == 1 ~ "Buick",
    Cadillac == 1 ~ "Cadillac",
    Chevy == 1 ~ "Chevy",
    Pontiac == 1 ~ "Pontiac",
    Saab == 1 ~ "Saab",
    Saturn == 1 ~ "Saturn"))) %>%
  mutate(Type = factor(case_when(
    coupe == 1 ~ "coupe",
    hatchback == 1 ~ "hatchback",
    sedan == 1 ~ "sedan",
    wagon == 1 ~ "wagon",
    TRUE ~ "NA"))) %>%
  select(!contains(brand.names) & !contains(car.types))

sat <- mycars %>%
  filter(Brand=="Saturn") %>%
  mutate(Type = fct_drop(Type))

sat <- subset(sat, select = -c(Brand, convertible))
```

2

Use train and method="lm" to compute a multiple linear regression model and get its cross-validated RMSE. Include the option savePredictions=TRUE in your trainControl open set.

A) Make sure to state your RMSE result in the text. This is the baseline RMSE that the other models should be able to beat.

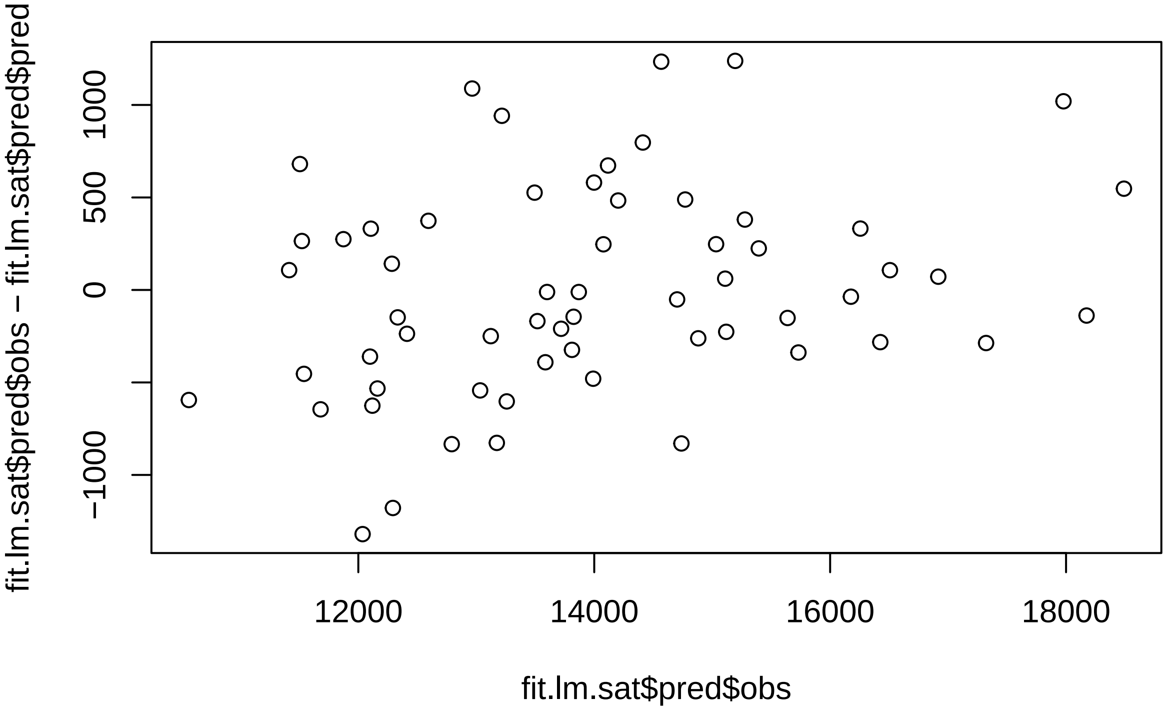
B) Also, make a plot of residuals vs. fit. This plot puts the difference between actual and predicted price on the y axis, plotted against actual price on the y axis.

For a well-fitting model, this cloud of residuals should be symmetric around the 0 line with no non-linear zigs or zags.

Comment in the text on how well your model appears to fit the data, based on this graph. You can use code that might look like this: plot(x=fit*pred*obs, y=fit*pred*obs-fit*pred*pred)

```
#need to add minimize RMSE function
fit.lm.sat <-
  train(Price ~ .,
        data=sat,
        method="lm",
        trControl=trainControl(method="repeatedcv",
                               number=100,
                               savePredictions="final",
                               classProbs=TRUE)
  )


#fit.lm.train$finalModel
plot(x=fit.lm.sat$pred$obs, y=fit.lm.sat$pred$obs-fit.lm.sat$pred$pred)
```



```
fit.lm.sat$results$RMSE
```

```
## [1] 449.2635
```

With 100-fold cross-validation, the RMSE was 449.2365.

3

Use train and method="svmRadialSigma" to tune a support vector machine. Include the option savePredictions=TRUE in your trainControl open set. As in Part 1 of the homework, give it a few tries to try to find optimal tuning parameters. Then do the following:

A) Plot RMSE vs. values of your tuning parameters and state in the text the values of the tuning parameters that optimized RMSE.

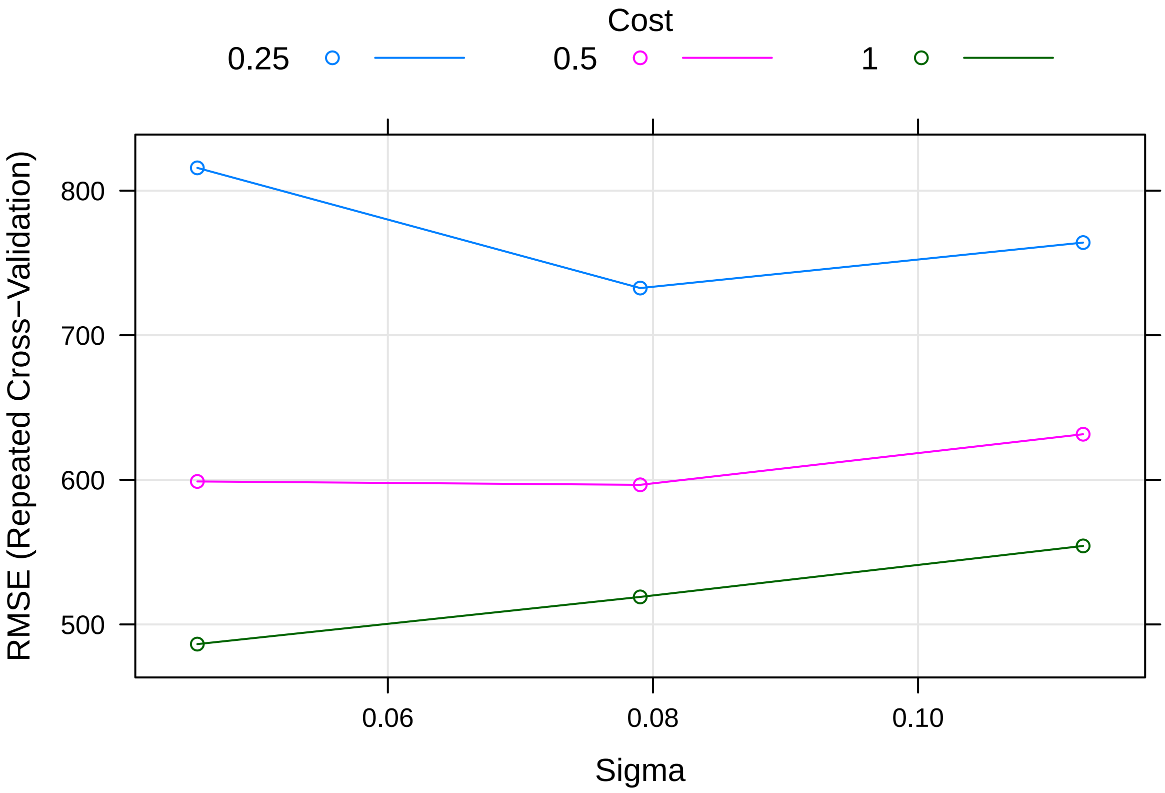
B) State the best RMSE you found in the text, and comment on whether it was higher or lower than the regular linear regression.

C) Make a plot of residuals vs. fit, as described above, for this model. Comment on how well the support vector machine seems to fit the data. Does it do better than the plan old linear regression?

```
fit.svm.sat <-
  train(Price ~ .,
        data=sat,
        method="svmRadialSigma",
        #maximize = ifelse(metric == "RMSE", FALSE, TRUE),
        trControl=trainControl(method="repeatedcv",
                               number=100,
                               repeats=10,
                               savePredictions="final"
                                )
  )
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

```
plot(fit.svm.sat)
```



```
fit.svm.sat$bestTune
```

```
##        sigma C
## 3 0.04562616 1
```

```
min(fit.svm.sat$results$RMSE)
```

```
## [1] 486.3909
```

The best RMSE using 100-fold cross-validation repeating 10 times was an RMSE of 486.1542. This is worse than linear regression RMSE.

Ideal sigma was .04389 with a cost of 1.

4

Do the same thing with a single-layer neural network using either the nnet or neuralnet method (caution: neuralnet runs slow, so start small).

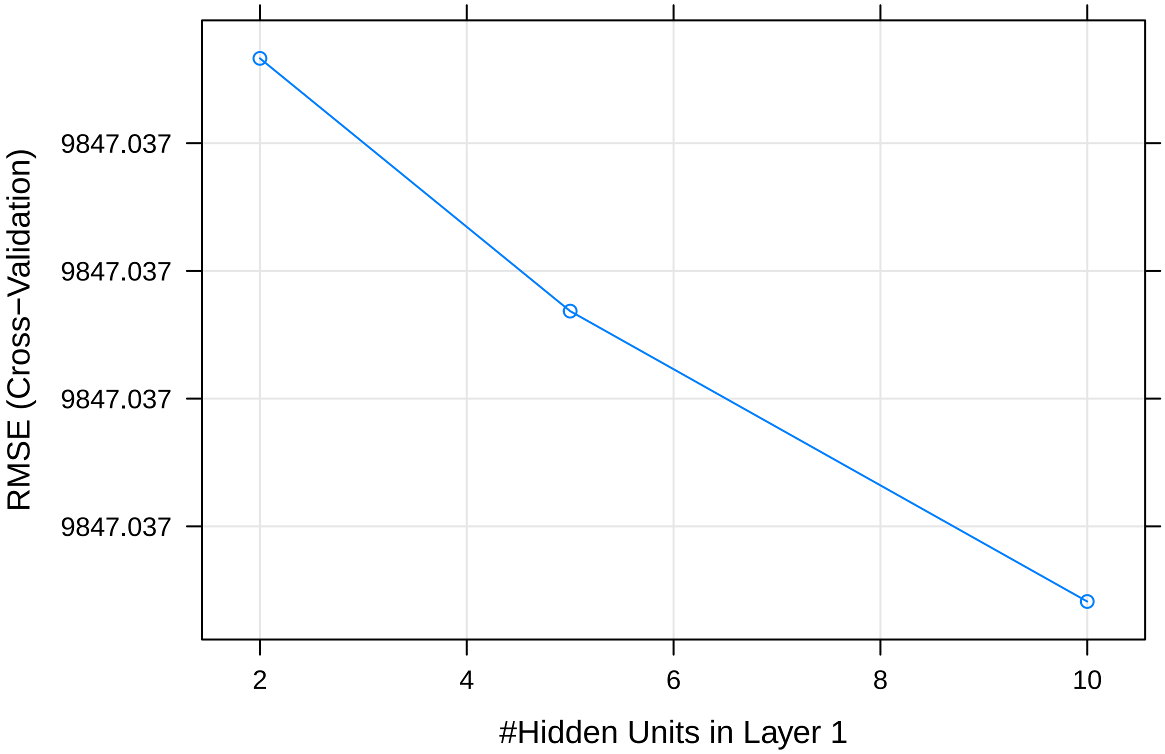
Remember the nnet is single-layer by design, and for neuralnet, you'll have to use tuning parameters to try different numbers of notes in layer1 and set the layer2 and layer3 parameters to zero.

Or, if you're feeling brave, look at one of the other neural network methods available through train. Neither nnet nor neuralnet seem to work very well with this data (perhaps because of all the 0/1 variables?), but comment on what you see. Or, maybe you'll find a way to make a neural net work!

```
fit.neuralnet.sat <-
  train(Price ~ Mileage + Type,
        data=mycars,
        method="neuralnet",
        trControl=trainControl(method="cv",
                               number=5,
                               savePredictions="final"),
        preProcess=c("scale", "center"),
        tuneGrid=expand.grid(layer1=c(2, 5, 10),
                             layer2=c(0), layer3=c(0)),
        threshold=0.3
  )
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

```
plot(fit.neuralnet.sat)
```



The best RMSE was nearly 10,000.

5

Finally, add a paragraph that sums it all up. Which method worked best?

Which fit the data best in this example?

The linear regression performed the best with a RMSE of 449. The SVM was close at 486 but given the added complexity, it was not a close 2nd. The neural net method had a RMSE of nearly 10,000. Significantly worse.

I went to research why neural nets performed poorly and found this post on stack exchange - https://stats.stackexchange.com/questions/240357/r-neuralnet-package-is-very-slow-and-the-final-results-are-not-good

Basically it has to do with how neural nets work and the inefficiencies associated with sparse data. Sparse data is basically a bunch of flat datapoints. Neural nets are searching for a minima so they are just randomly wandering from point to point trying to find a minima. This explains their weakness as well as the time it takes to process.

In my eyes, it's a flat line with a random spike here or there. The goal is to find the lowest spike.

A neural net works much better when dealingwith data that allows for more data at each point, able to treat the data as a gradient and working the dataframe like a parabola, quickly finding a minima.

This makes sense when looking at the neural net activation function - which either follows a step-like function or a Gaussian distribution per this source: https://www.quora.com/Why-are-deep-neural-networks-so-bad-with-sparse-data I probably should look at the neuralnet source code but based off the warning "Algorithm did not converge within the stepmax" I will state neural net utilizes a step-like function

I found a paper from October 2020 that basically summed up neural nets as an area of further research for optimization problems with sparse data - which I take as meaning there is no solution/reason for applying neural nets to sparse data.