# STP 598: Homework 5

*Antonio Campbell, Sinta Sulistyo, Atta Ullah, Penny Wu*

*4/12/2021*

### Problem 1

We upload the used cars data first and then generate a subset of the actual dataset which has only two features price and mileage. We rescale both the features dividing by 1000 and replace price with variable name $y$ and mileage with the variable name $x$.

```
cd1 = read.csv("http://www.rob-mcculloch.org/data/usedcars.csv")
cd2 = cd1[,c(1,4)];
cd = cd1[, c(1,4)]/1000;
names(cd)[names(cd) == "price"] <- "y"
names(cd)[names(cd) == "mileage"] <- "x"
```

### Train, Validation and Test Split

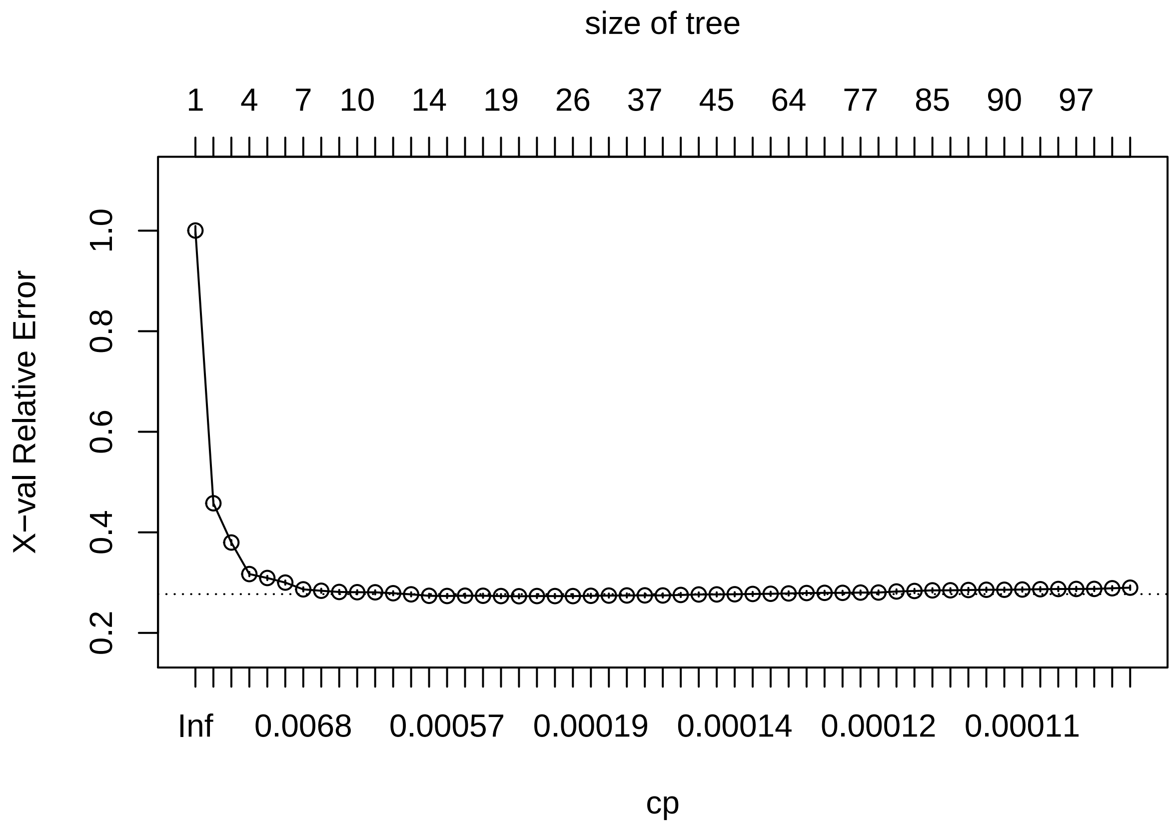
In order to use the three set approach, we divide the data into three sets, the train set set, the validation set and the test set, as the following:

```
set.seed(1722)
n=nrow(cd)
n1=floor(n/2)
n2=floor(n/4)
n3=n-n1-n2
ii = sample(1:n,n)
cdtrain = cd[ii[1:n1],]
cdval = cd[ii[n1+1:n2],]
cdtrainval = rbind(cdtrain,cdval)
cdtest = cd[ii[n1+n2+1:n3],]
```

### Trees

We fit a big tree on the training data.

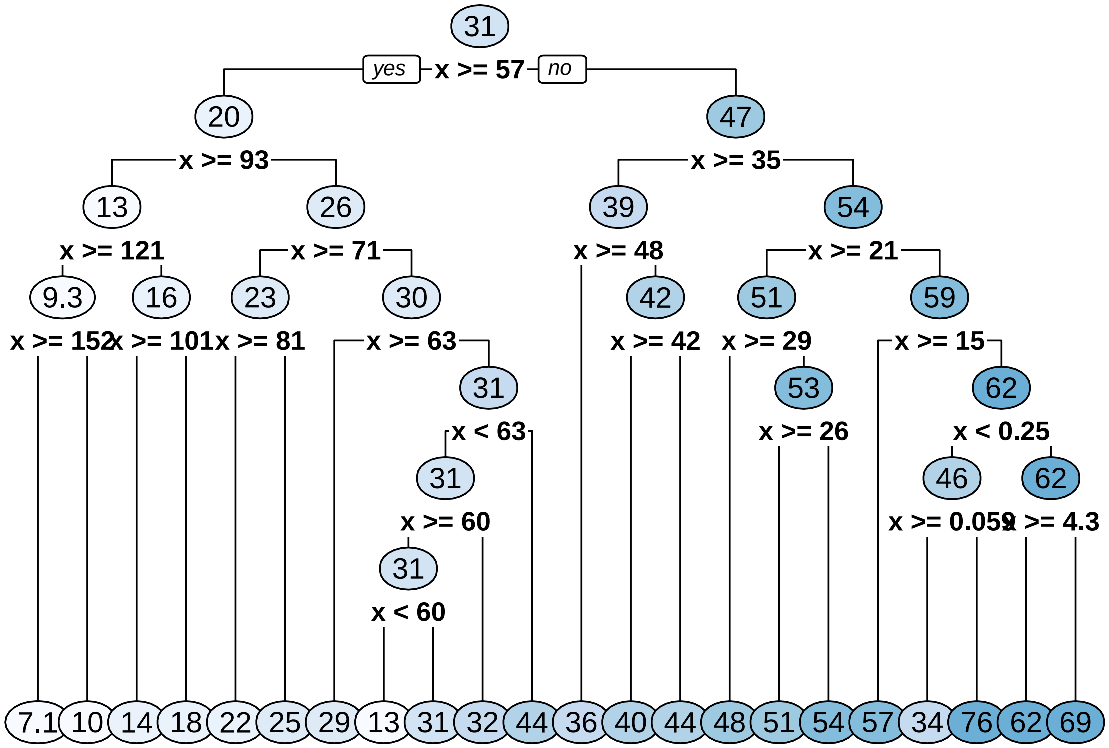
```
## Size of big tree:  109
```



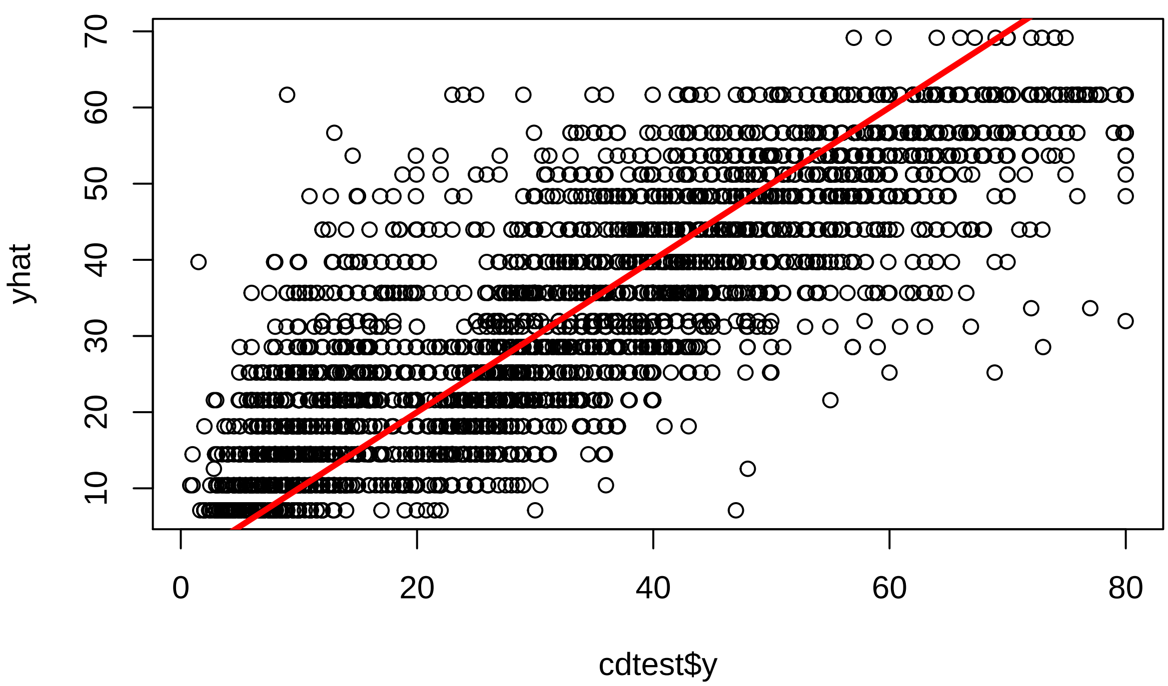

```r
# get nice tree from CV results
iibest = which.min(big.tree$cptable[,"xerror"]) #which has the lowest error
bestcp=big.tree$cptable[iibest,"CP"]
bestsize = big.tree$cptable[iibest,"nsplit"]+1
```

We found best `cp` and then prune the tree that gives us trees of various sizes. We now make prediction on the validation data based on the pruned tree:

```
## Size of best tree:  22
```




Provided are the fits for this model:

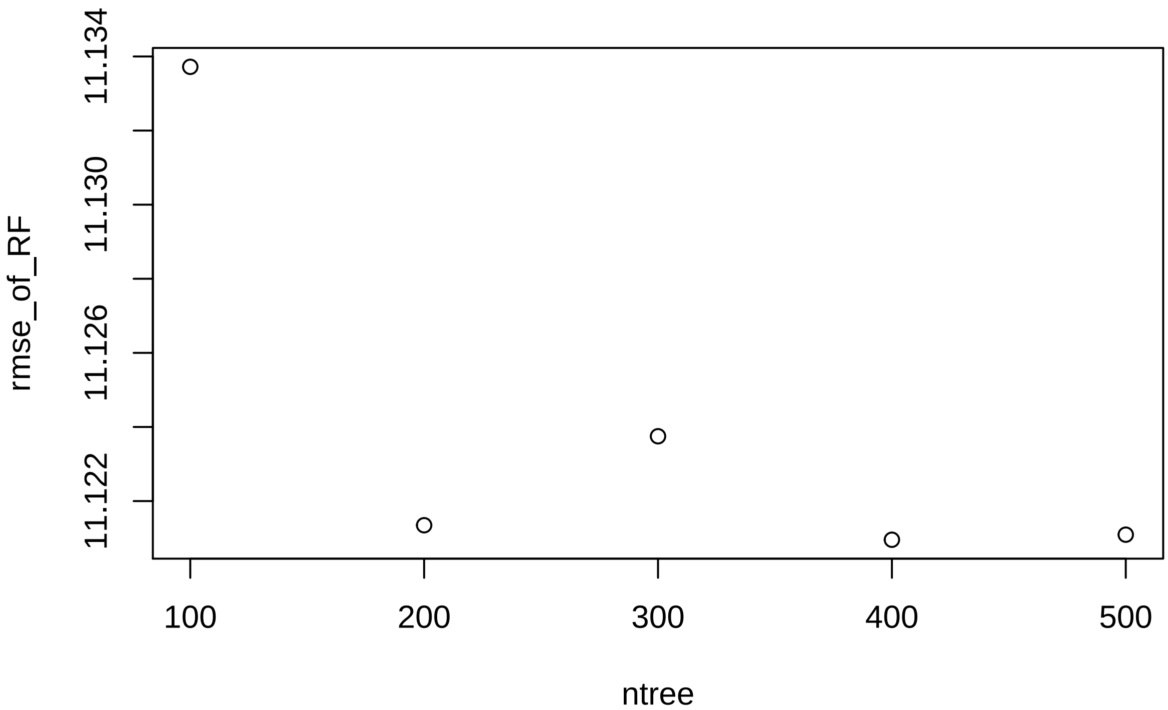



**Random Forests**

We fit five different random forests to the training data with different numbers of trees. In this case we take $n = 100, 200, 300, 400, 500$. Since we only have 1 variable $x$, the dimension of $x$ is 1. This means the `mtry` $= 1$.

Now use all these fitted models to predict over the validation data.

Calculating the RMSE of each model:




We see that 400 trees give us lowest root mean squared error on the validation dataset. We refit a random forest using 400 trees on the union of the train and validation data and then later measure the accuracy over the test data.

```
rffit6 = randomForest(y ~ x, data = cdtrainval, mtry = 1, ntree =minN)
rfvalpred6 = predict(rffit6, newdata = cdtest)
rmse6 = sqrt(mean((cdtest$y-rfvalpred6)^2))
print(rmse6)
```

```
## [1] 10.8001
```

We successfully used random forest to predict the price of used cars using mileage as the prediction variable.

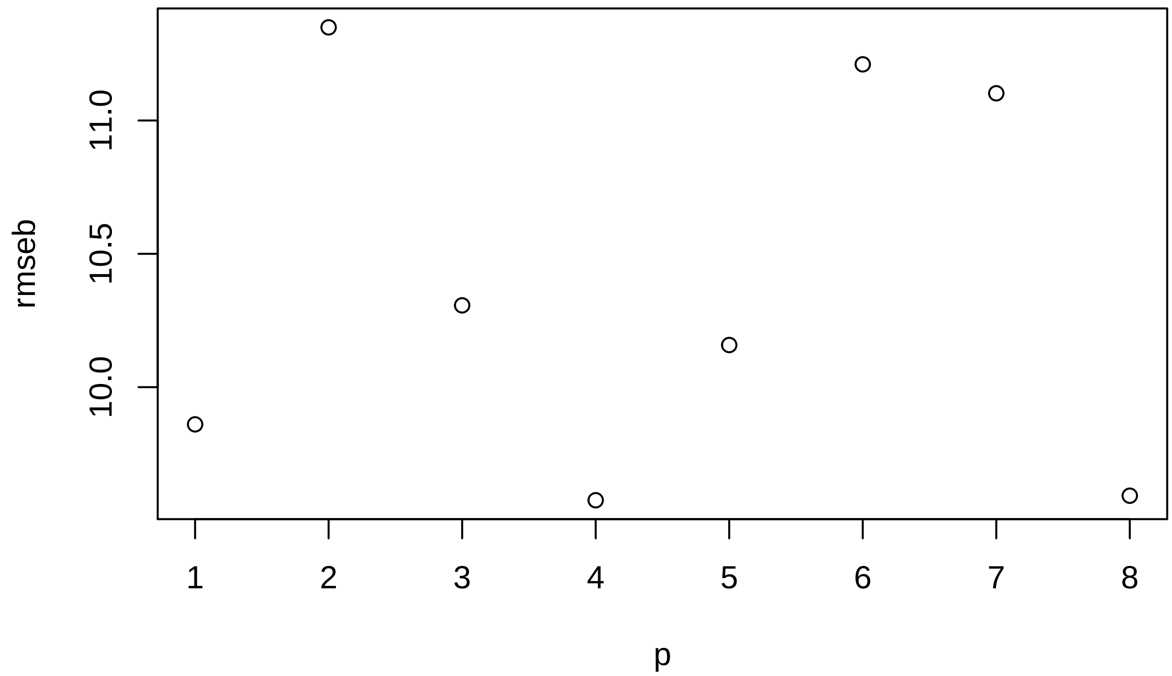
### Boosting

We now use boosting to fit the relate the price and the mileage. We use 8 different combinations of parameters to fit the models on the test date and chose the one that gives best prediction on the validation data. The eight combinations are the following: ##### Maximum depth of 4 with 1000 trees and $\lambda = 0.2$ ##### Maximum depth of 4 with 1000 trees and $\lambda = 0.001$ ##### Maximum depth of 4 with 5000 trees and $\lambda = 0.2$ ##### Maximum depth of 4 with 5000 trees and $\lambda = 0.001$ ##### Maximum depth of 10 with 1000 trees and $\lambda = 0.2$ ##### Maximum depth of 10 with 1000 trees and $\lambda = 0.0.001$ ##### Maximum depth of 10 with 5000 trees and $\lambda = 0.2$ ##### Maximum depth of 10 with 5000 trees and $\lambda = 0.001$

```
boostfit1 = gbm(y~x,data = cdtrain,distribution="gaussian",
interaction.depth = 4, n.trees = 1000,shrinkage = 0.2)
boostfit2 = gbm(y~x,data=cdtrain,distribution = "gaussian",
interaction.depth = 4, n.trees = 1000,shrinkage = 0.001)
boostfit3 = gbm(y~x,data=cdtrain,distribution = "gaussian",
interaction.depth =4, n.trees = 5000,shrinkage = 0.2)
boostfit4 = gbm(y~x,data=cdtrain,distribution = "gaussian",
interaction.depth=4, n.trees = 5000,shrinkage = 0.001)
```

```r
boostfit5 = gbm(y~x,data=cdtrain,distribution="gaussian",
interaction.depth = 10, n.trees = 1000,shrinkage = 0.2)
boostfit6 = gbm(y~x,data=cdtrain,distribution = "gaussian",
interaction.depth = 10, n.trees = 1000,shrinkage = 0.001)
boostfit7 = gbm(y~x,data=cdtrain,distribution = "gaussian",
interaction.depth = 10, n.trees = 5000,shrinkage = 0.2)
boostfit8 = gbm(y~x,data=cdtrain,distribution = "gaussian",
interaction.depth = 10, n.trees = 5000,shrinkage = 0.001)
```

We now make prediction on the validation data based on each fit and calculate the RMSE.




We see that the minimum root mean squared corresponds to the combination 4 which corresponds to the fourth model where we used 5000 trees with depth 4 and $\alpha = 0.001$. We now use boosting again on the union of training and validation data and predict on the test data with 5000 trees with depth 4 and $\alpha = 0.001$.

```r
boostfit = gbm(y~x,data = cdtrainval ,distribution="gaussian",
interaction.depth = 4, n.trees = 5000,shrinkage = 0.001)
boostvalpred = predict(boostfit, newdata = cdtest, n.trees = 5000)
rmseb = sqrt(mean((cdtest$y - boostvalpred)^2))
print(rmseb)
```

```
## [1] 9.500034
```

We successfully used the boosting to predict the price of used cars using mileage as the prediction variable.

### Problem 3

Use a neural net to relate $y =$ `price` to $x =$ `mileage`. Use the three set approach, that is, split your data into train, validation, test set. Plot your results.

### Solution:

We first re-extract a subset of the dataset with only two columns, containing price and mileage and then rescale the price as can be seen in the following:

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       8   39888   67187   73114   98213  488525
```

```
##      price          mileage
##  Min.   :  599   Min.   :0.00000
##  1st Qu.:13495   1st Qu.:0.08164
##  Median :29454   Median :0.13752
##  Mean   :30747   Mean   :0.14965
##  3rd Qu.:43995   3rd Qu.:0.20103
##  Max.   :79999   Max.   :1.00000
```

### Train, Validation and Test Split

```
set.seed(224)
n = nrow(cd2)
n1 = floor(n/2)
n2 = floor(n/4)
n3 = n-n1-n2
ii = sample(1:n,n)
cdtrain_nn = cd2[ii[1:n1],]
cdval_nn = cd2[ii[n1+1:n2],]
cdtrainval_nn = rbind(cdtrain_nn,cdval_nn)
cdtest_nn = cd2[ii[n1+n2+1:n3],]
```

### Different fits with different size and decay parameters:

Now we fit different single layer neural nets on the training data with `size` $= 25, 75$ and `decay`$= 0.5, 0.01$. We choose the sizes 25 and 75 since the number of the hidden units is usually in the range of 5 to 100.

```
nn_fit1 = nnet(price ~ mileage, cdtrain_nn, size = 25, decay = 0.5, linout=T)
```

```
## # weights:  76
## initial  value 12807168527619.957031
## iter  10 value 3373902033761.175293
## iter  20 value 3252412060691.823242
## iter  30 value 1546549761133.804932
## iter  40 value 1452042248367.736328
## iter  50 value 1294205209177.035400
## iter  60 value 1234874243211.430420
## iter  70 value 1195001068558.287598
## iter  80 value 1182977556902.599854
## iter  90 value 1159399667330.253662
## iter 100 value 1075584608779.434570
## final  value 1075584608779.434570
## stopped after 100 iterations
```

```r
nn_fit2 = nnet(price ~ mileage, cdtrain_nn, size = 25, decay= 0.01, linout=T)
```

```
## # weights:  76
## initial  value 12807434785771.216797
## final  value 3373803945980.632812
## converged
```

```r
nn_fit3 = nnet(price ~ mileage, cdtrain_nn, size = 75, decay = 0.5, linout=T)
```

```
## # weights:  226
## initial  value 12808334783269.843750
## iter  10 value 1551356159629.211670
## iter  20 value 1536904558049.150635
## iter  30 value 1427504493714.726318
## iter  40 value 1268095564152.181396
## iter  50 value 1233827427891.860596
## iter  60 value 1122668498929.165771
## iter  70 value 1076624143582.124390
## iter  80 value 1067559609548.587280
## iter  90 value 1054923481341.825928
## iter 100 value 1036067101082.700562
## final  value 1036067101082.700562
## stopped after 100 iterations
```

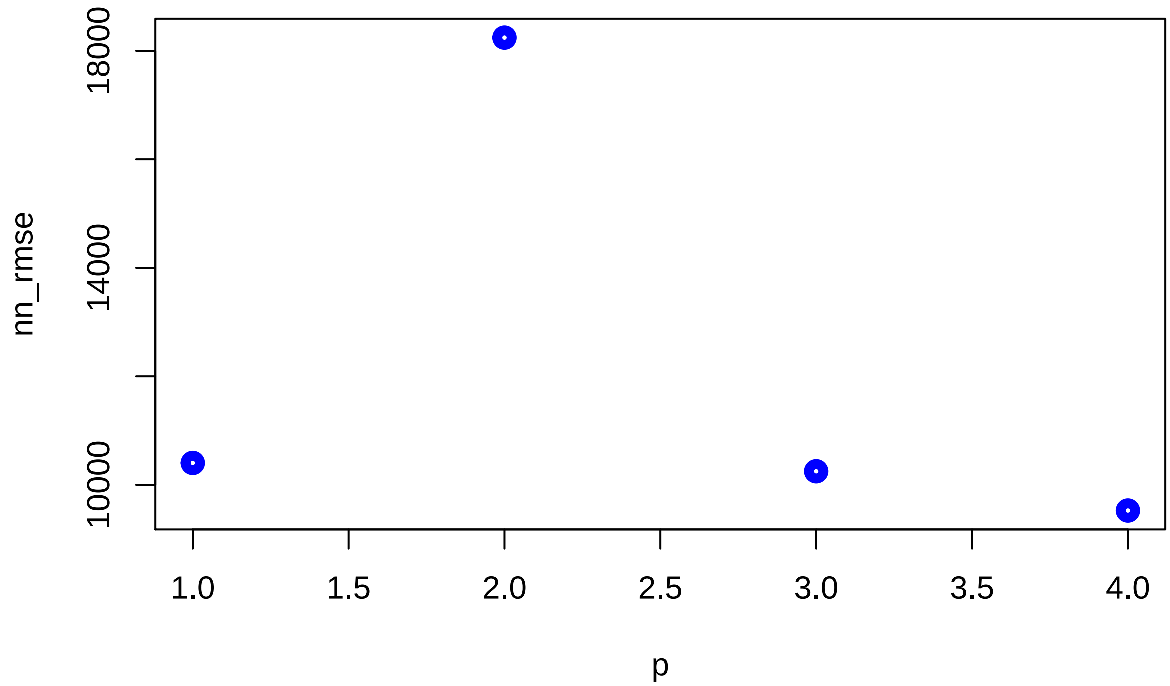
```r
nn_fit4 = nnet(price ~ mileage, cdtrain_nn, size  = 75, decay = 0.01, linout = T)
```

```
## # weights:  226
## initial  value 12808281138711.169922
## iter  10 value 1882782298061.946533
## iter  20 value 1526343326318.410400
## iter  30 value 1522713886659.230957
## iter  40 value 1294410690799.809814
## iter  50 value 1191999450701.334961
## iter  60 value 1119054214946.233887
## iter  70 value 1077581096556.959595
## iter  80 value 968625778073.281372
## iter  90 value 904698431314.584351
## iter 100 value 901326864415.802734
## final  value 901326864415.802734
## stopped after 100 iterations
```

**Predictions on the Validation set:**

```r
temp1 = data.frame(price = cdval_nn$price, mileage = cdval_nn$mileage)
nn_predict1 = predict(nn_fit1, temp1)
nn_predict2 = predict(nn_fit2, temp1)
nn_predict3 = predict(nn_fit3, temp1)
nn_predict4 = predict(nn_fit4, temp1)
```

We now calculate the loss function for each prediction.




```r
which.min(nn_rmse)
```

```
## [1] 4
```

Thus the root mean squared error corresponding to the third fit is minimum of the four which has `size` $= 75$ and `decay` $= 0.01$.

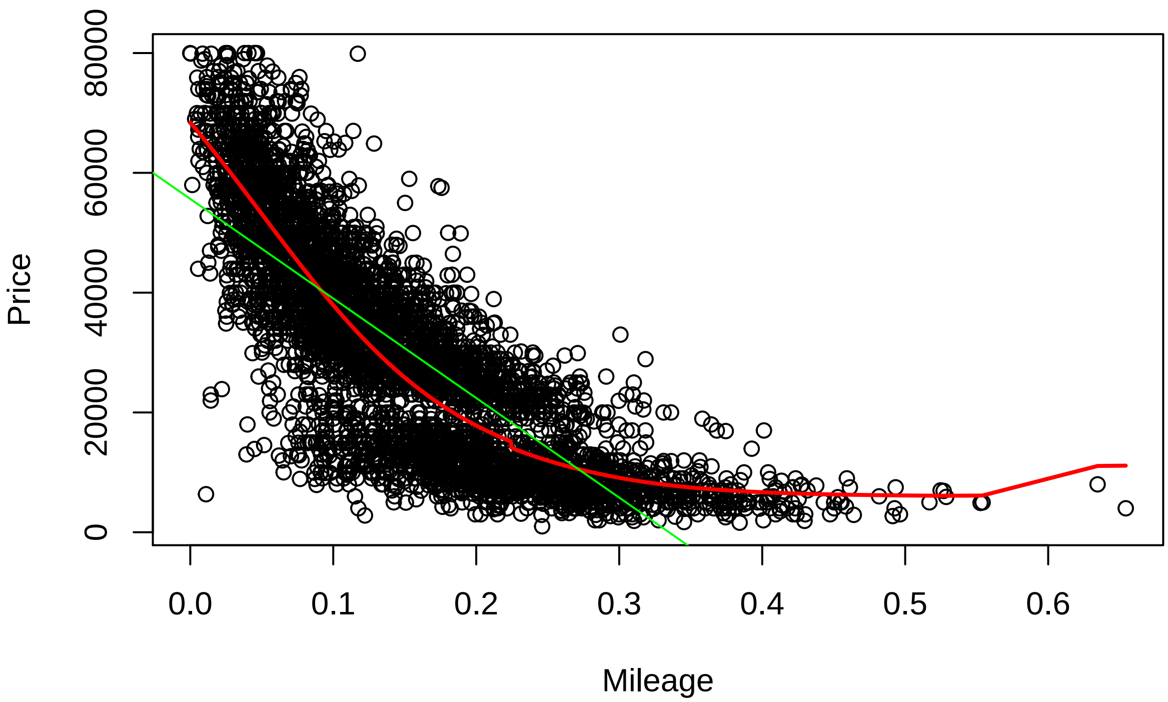
**Fit on the union of Train and Validation data:**

We now fit a single layer neural net with size $= 75$ and decay $= 0.01$ on the train validation set and the predict on the test data.

```
## # weights:  226
## initial  value 19276630299757.144531
## iter  10 value 3935313069564.791504
## iter  20 value 2224389652669.733398
## iter  30 value 1701693280061.425537
## iter  40 value 1369671437983.673096
## iter  50 value 1353514505147.837402
## iter  60 value 1352319001161.256592
## iter  70 value 1348939616384.102295
## iter  80 value 1348451353969.607178
## iter  90 value 1348405803087.188965
## iter 100 value 1348000944638.870117
## final  value 1348000944638.870117
## stopped after 100 iterations
```

```r
print(nn_loss6)
```

```
## [1] 9523.97
```




**Problem 4**

We continue from the previous example by adding a second predictor into our neural net model. We have `x1 = mileage` and `x2 = year`. We rescale `year` as we did with `mileage`:

```
dat = cbind(cd2, year = ((cd1$year-min(cd1$year))/(max(cd1$year)-min(cd1$year))))
summary(dat$year)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.5000  0.6500  0.6505  0.8000  1.0000
```

We partition the train, validation, and test data using the same indices as before for comparison of the fits:

```
cdtrain_nn = dat[ii[1:n1],]
cdval_nn = dat[ii[n1+1:n2],]
cdtrainval_nn = rbind(cdtrain_nn,cdval_nn)
cdtest_nn = dat[ii[n1+n2+1:n3],]
```

We again will fit a simple model with various values of `decay` and `size` using a grid of combinations of both.

```
val = data.frame(price = cdval_nn$price, mileage = cdval_nn$mileage, year = cdval_nn$year)
grid = expand.grid(size = c(5, 25, 50, 75, 100), decay = c(0.001 , 0.01, 0.05 ,0.1, 0.25, 0.5))
grid1 = cbind(grid, rmse = 1)

for(i in 1:nrow(grid)){
  fit = nnet(price ~ mileage+year, cdtrain_nn, size = grid[i,1], decay = grid[i,2], linout=T, trace = FA
  pred = predict(fit, val)
  grid1[i,3] = rmse(val$price,pred)
}
```

For this particular grid search the RMSE is minimized when the value of `size` is 75 and the the `decay` is 0.05.

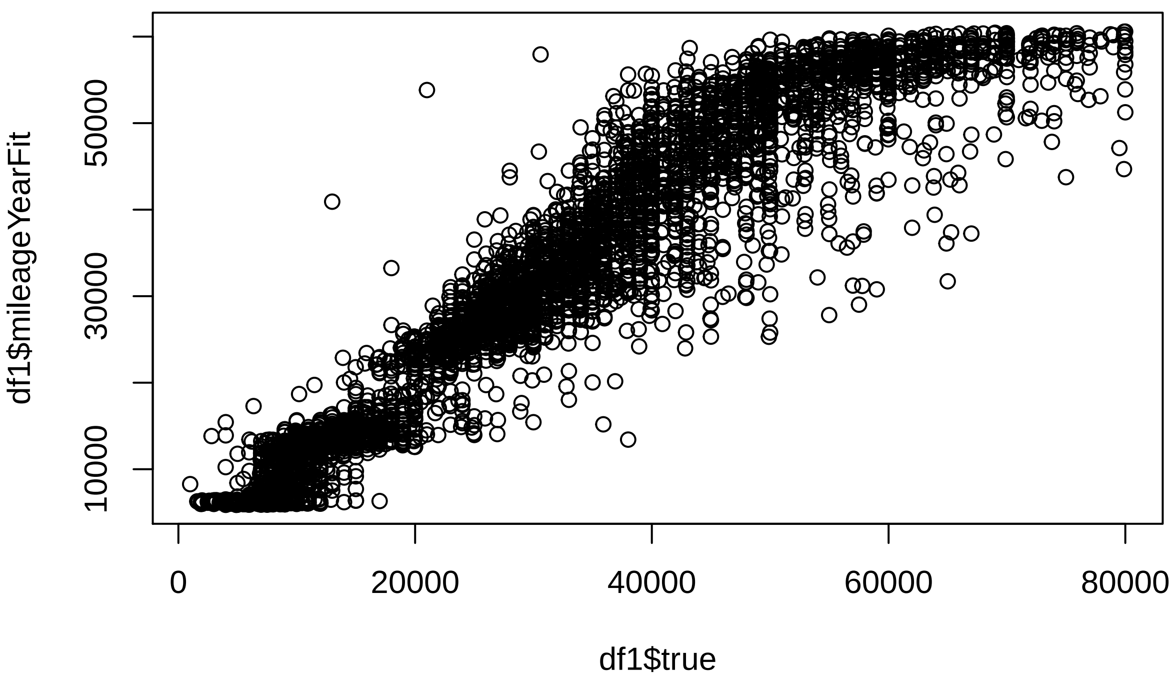
Now let's see what the fits look like on our test data.

```r
ind = which(grid1$rmse==min(grid1$rmse))
tval = data.frame(price = cdtrainval_nn$price, mileage = cdtrainval_nn$mileage, year = cdtrainval_nn$yea
good_fit = nnet(price ~ mileage+year, cdtrainval_nn, size = grid1[ind,1] , decay = grid1[ind,2], linout
```

```
## # weights:  301
## initial  value 19273457122469.695312
## iter  10 value 2825206061578.372070
## iter  20 value 1263881991554.251221
## iter  30 value 1145519221535.440674
## iter  40 value 1054572510625.818481
## iter  50 value 946005888982.922607
## iter  60 value 903260405396.541260
## iter  70 value 724319029727.784302
## iter  80 value 537688117014.301147
## iter  90 value 470114765881.446777
## iter 100 value 466571799144.775513
## final  value 466571799144.775513
## stopped after 100 iterations
```

```r
test = data.frame(price = cdtest_nn$price, mileage = cdtest_nn$mileage, year = cdtest_nn$year)
pred = predict(good_fit, test)
rmse = rmse(test$price, pred)
```

The RMSE for the test data at our best fit is 5589.0615081. Let's compare the models optimized by a simple grid search in problems 3 and 4.




```
##                      true mileageYearFit mileageFit
## true            1.0000000      0.9519574  0.8530142
## mileageYearFit  0.9519574      1.0000000  0.8872434
## mileageFit      0.8530142      0.8872434  1.0000000
```

We can see an improvement in model fit once we introduce another variable.