

Homework 1 Solution

Question 1

Here is a one possible solution.

We first set up parameters and load relevant libraries.

```
rm(list=ls())
library(data.table)
library(kknn)

# Primitives
N_train=100
N_test=10000
```

Next, we write functions for generating data and plotting.

```
# Function to simulate data
Simulate <- function(func, N_train, N_test){
  set.seed(11)
  n=N_train+N_test
  x=runif(n, -1, 1)
  y=func(x)+rnorm(n, mean=0, sd=0.15)
  dt=data.table(y=y,x=x)
  tr_idx=sample(1:n, N_train)
  train=dt[tr_idx,]
  setkey(train,x)
  test=dt[-tr_idx,]
  setkey(test,x)
  return(list(model=func, full=dt, train=train, test=test))
}

# Function to create base scatter plot
Plot <- function(sim, method){
  # unpack
  true_model=sim$model
  train=sim$train
  if (method=="linear regression"){
    plot(train$x, train$y, cex=0.5, pch=19, col="grey")
    title(main="Scatter plot, true relationship, and linear fit")
    lines(train$x, true_model(train$x), col="black", lwd=2)
    linear=lm(y~x, data=train)
    yhat=predict(linear, train)
    lines(train$x, yhat, col="blue", lty=2, lwd=3)
    return(linear)
  } else if (method=="k-nn") {
    k_vec=2:15
    #near_list <- vector("list", length(k_vec))
    fit=matrix(0, nrow=dim(train)[1], ncol=length(k_vec))
    for (i in 1:length(k_vec)){
      k=k_vec[i]
      near <- kknn(y~x, train=train, test=train[,.(x)], k=k, kernel='rectangular')
```

```

    #near_list[[i]]<-near
    fit[,i]<-near$fitted
  }

  par(mfrow=c(1,2))
  plot(train$x, train$y, cex=0.5, pch=19, col="grey")
  title(main="k-nn fit: k=2")
  lines(train$x, true_model(train$x), col="black", lwd=2)
  lines(train$x, fit[,1], col="blue", lwd=2)
  plot(train$x, train$y, cex=0.5, pch=19, col="grey")
  title(main="k-nn fit: k=12")
  lines(train$x, true_model(train$x), col="black", lwd=2)
  lines(train$x, fit[,11], col="blue", lwd=2)
  return(k_vec)
}
}

# Function to plot test set MSE
PlotTestError <- function(sim, base_plot_ls, base_plot_knn){
  linear=base_plot_ls
  k_vec=base_plot_knn
  train=sim$train
  test=sim$test
  testMSE_vec=numeric(length=length(k_vec))
  for (i in 1:length(k_vec)){
    near=kknn(y~x, train=train, test=test[,.x], k=k_vec[i], kernel='rectangular')
    testMSE_vec[i]=mean((near$fitted-test$y)^2)
  }
  lsMSE=mean((predict(linear, test)-test$y)^2)
  par(mfrow=c(1,1))
  plot(log(1/k_vec), testMSE_vec,
       ylim = c(0.95*min(c(testMSE_vec, lsMSE)), 1.05*max(c(testMSE_vec, lsMSE))),
       type="n")
  title(main="Test set mean squared error")
  lines(log(1/k_vec), testMSE_vec, lty=6, lwd=3, col="forestgreen", type="o", cex=0.6 )
  abline(h=lsMSE, col="black", lty=2)
}

```

Linear model (part 1.1 - 1.5)

Define function

```
func_linear <- function(x){1.8*x+2}
```

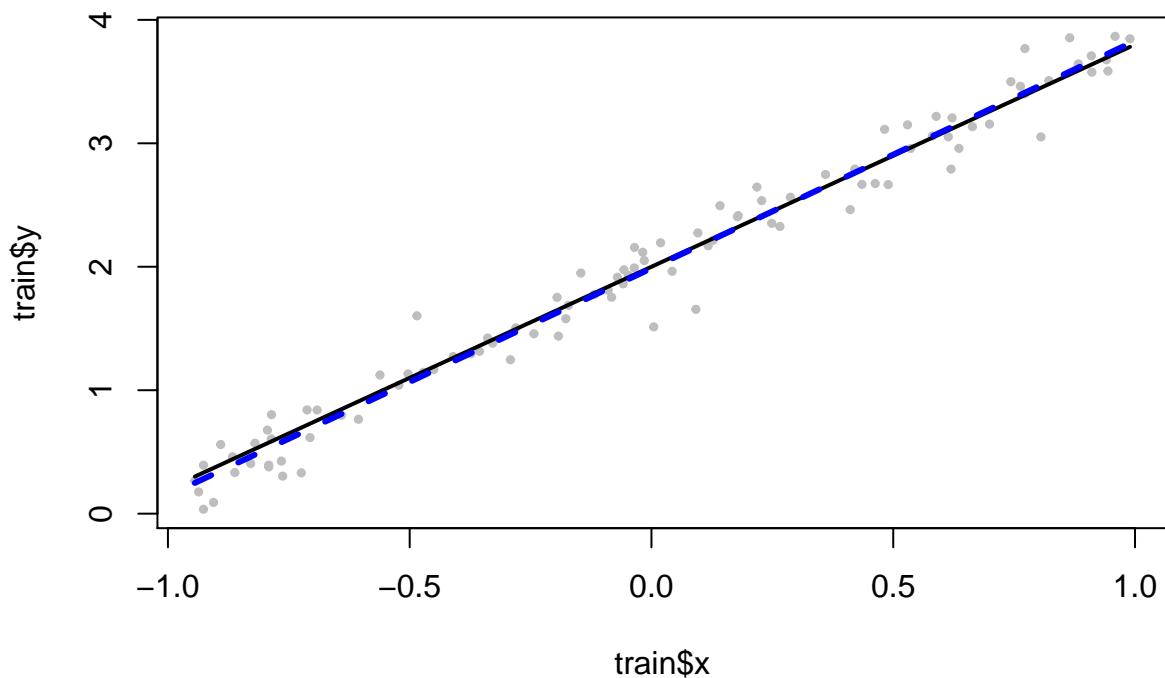
Simulate data from the linear function

```
sim_linear <- Simulate(func_linear, N_train, N_test)
```

Plot the training dataset, true relationship and best linear fit

```
plot_ls<- Plot(sim_linear, method = "linear regression")
```

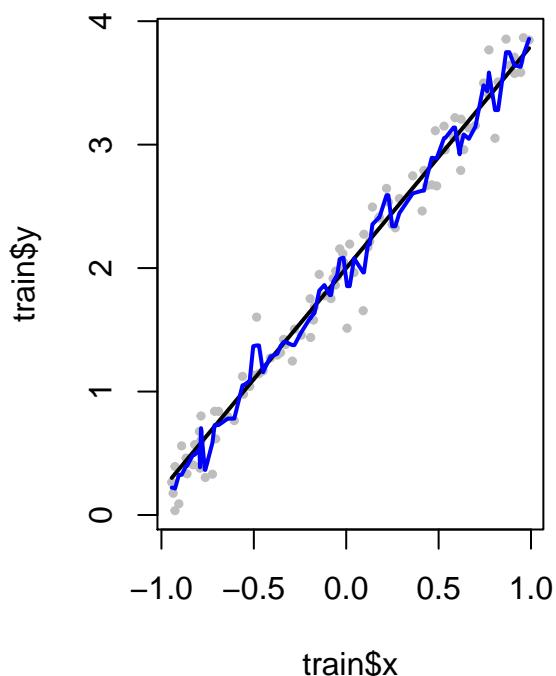
Scatter plot, true relationship, and linear fit



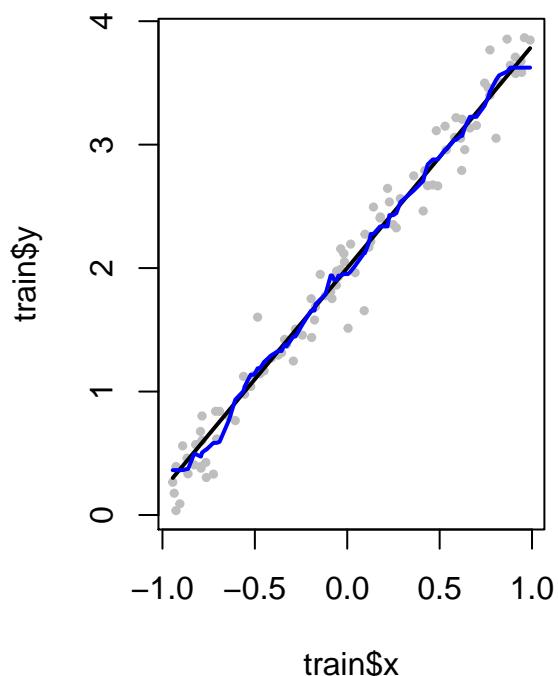
Plot k-NN fit with k=2 and k=12 respectively

```
plot_knn <- Plot(sim_linear, method="k-nn")
```

k-nn fit: k=2



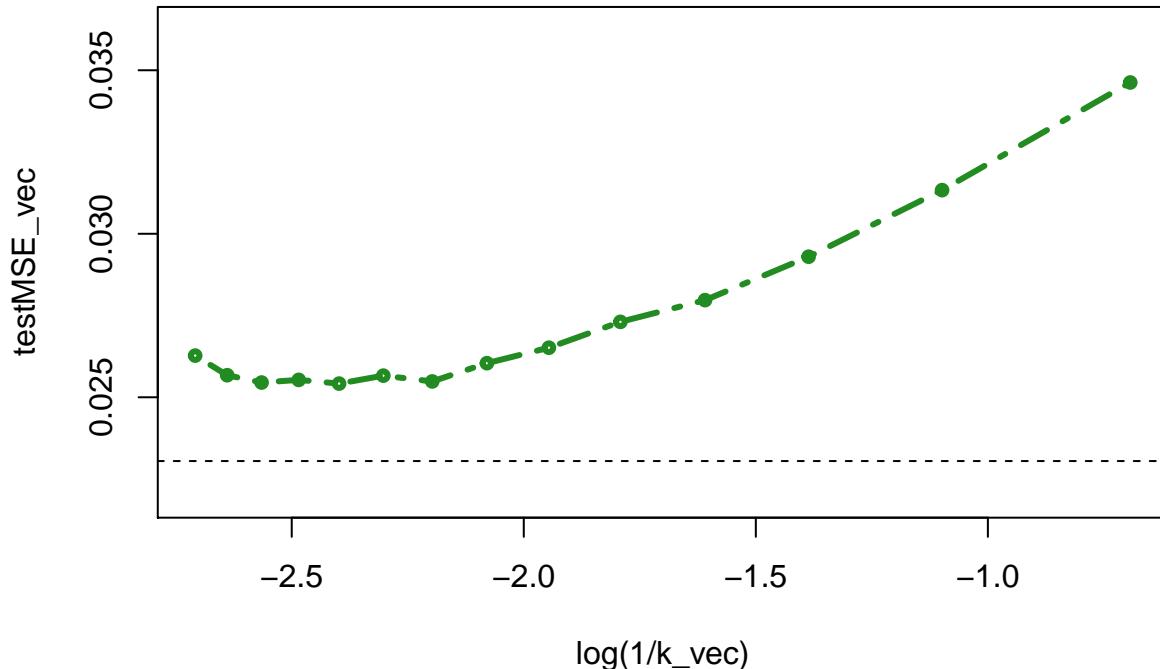
k-nn fit: k=12



Plot test set MSE

```
plotMSE <- PlotTestError(sim_linear, plot_ls, plot_knn)
```

Test set mean squared error



- Observation

Linear regression model outperforms k-NN algorithm for any value of k . In fact, when the true relationship is linear, it is hard for a non-parametric approach to compete with linear regression. Also notice that when the value of k is large, i.e. the model is simple, k-NN performs only a little worse than least square regression. It does far worse when k is small.

Almost linear model (part 1.6)

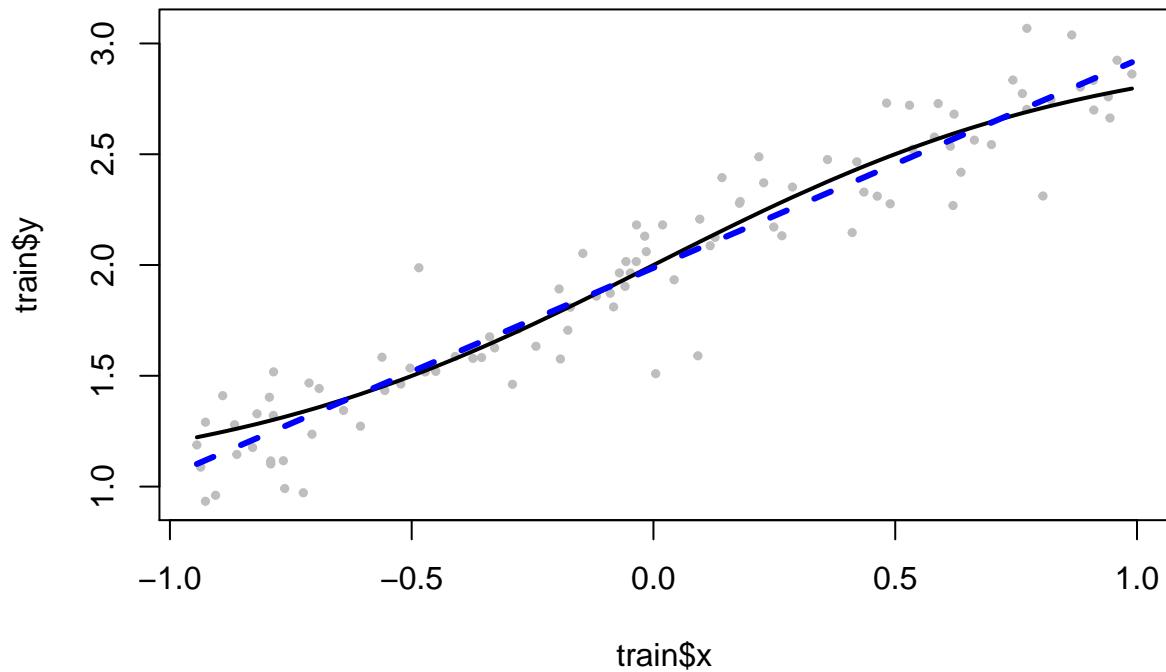
Define function

```
func_almost_linear <- function(x){tanh(1.1*x)+2}
```

Simulate data and plot

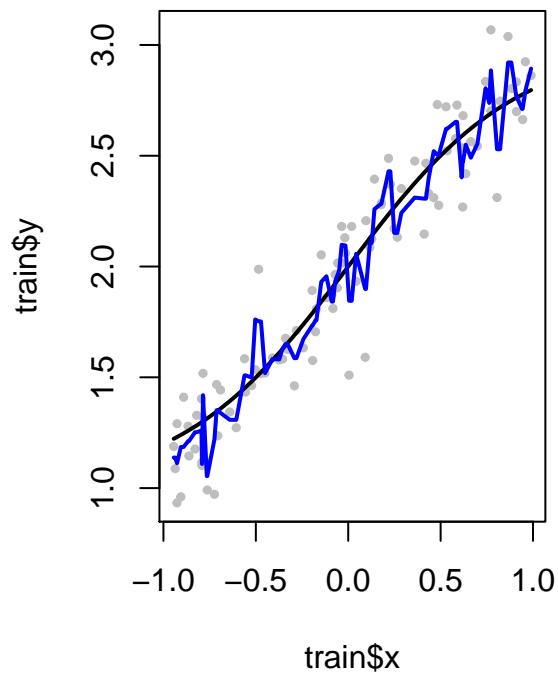
```
sim_aslinear <- Simulate(func_almost_linear, N_train, N_test)
plot_ls_aslinear <- Plot(sim_aslinear, method='linear regression')
```

Scatter plot, true relationship, and linear fit



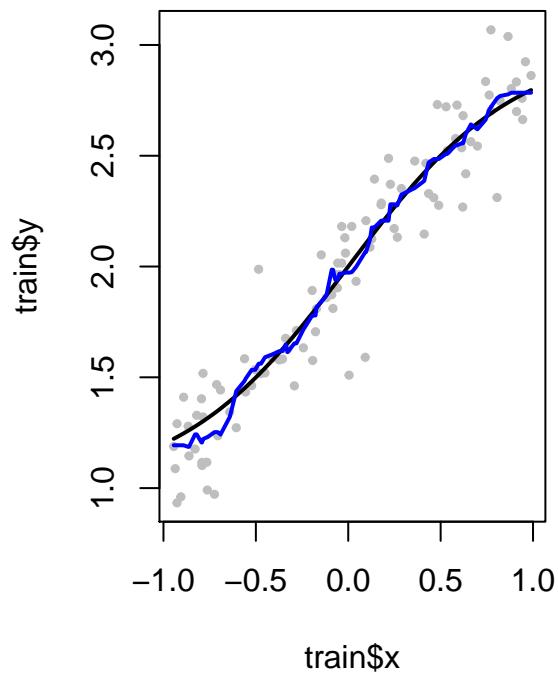
```
plot_knn_aslinear <- Plot(sim_aslinear, method='k-nn')
```

k-nn fit: k=2

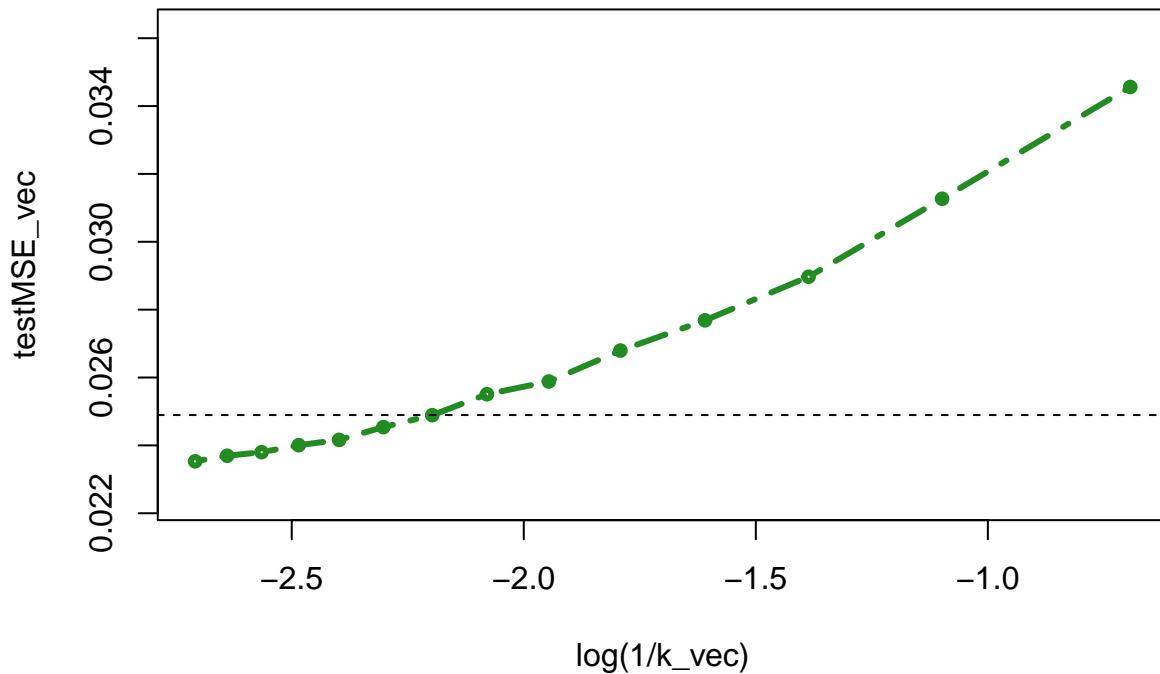


```
plotMSE_aslinear <- PlotModelError(sim_aslinear, plot_ls_aslinear, plot_knn_aslinear)
```

k-nn fit: k=12



Test set mean squared error



- Observation

The test MSE for linear regression is superior to that of k-NN for low values of k . However, for larger k , k-NN outperforms linear regression.

Strongly nonlinear model (part 1.7)

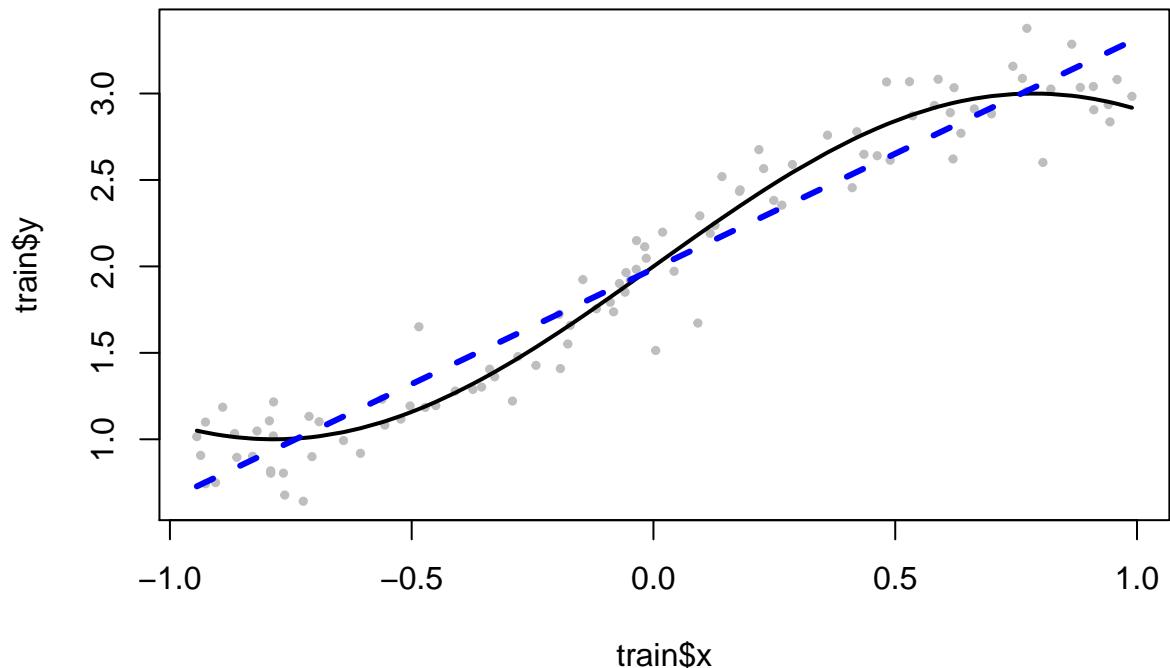
Define function

```
func_non_linear <- function(x){sin(2*x)+2}
```

Simulate data and plot

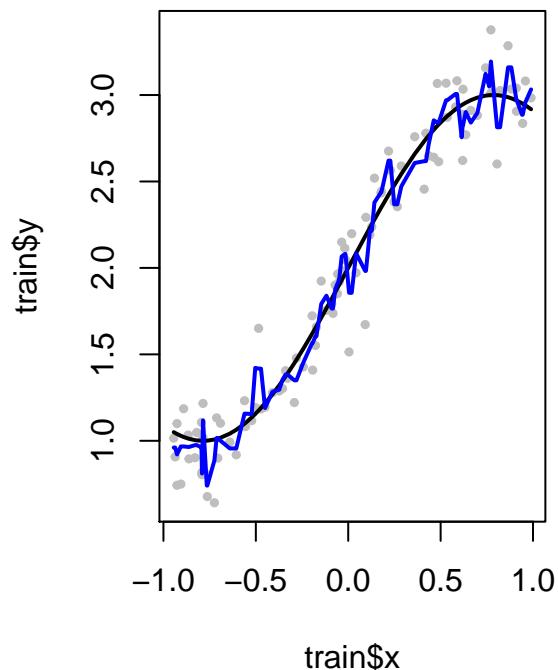
```
sim_nlinear <- Simulate(func_non_linear, N_train, N_test)
plot_ls_nlinear <- Plot(sim_nlinear, method='linear regression')
```

Scatter plot, true relationship, and linear fit

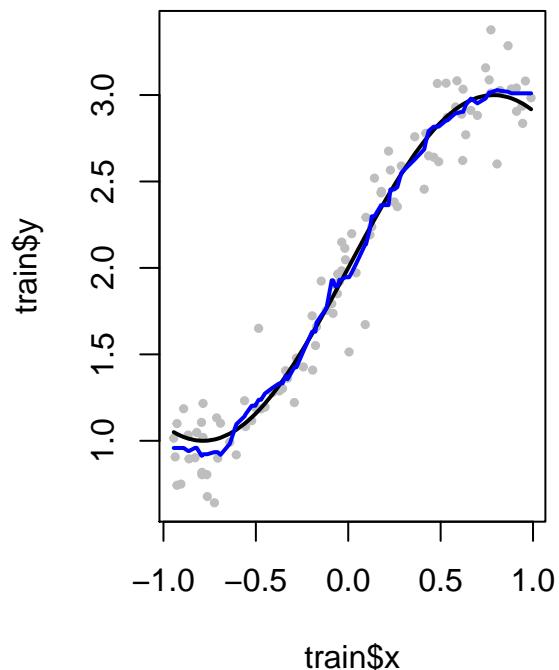


```
plot_knn_nlinear <- Plot(sim_nlinear, method='k-nn')
```

k-nn fit: k=2

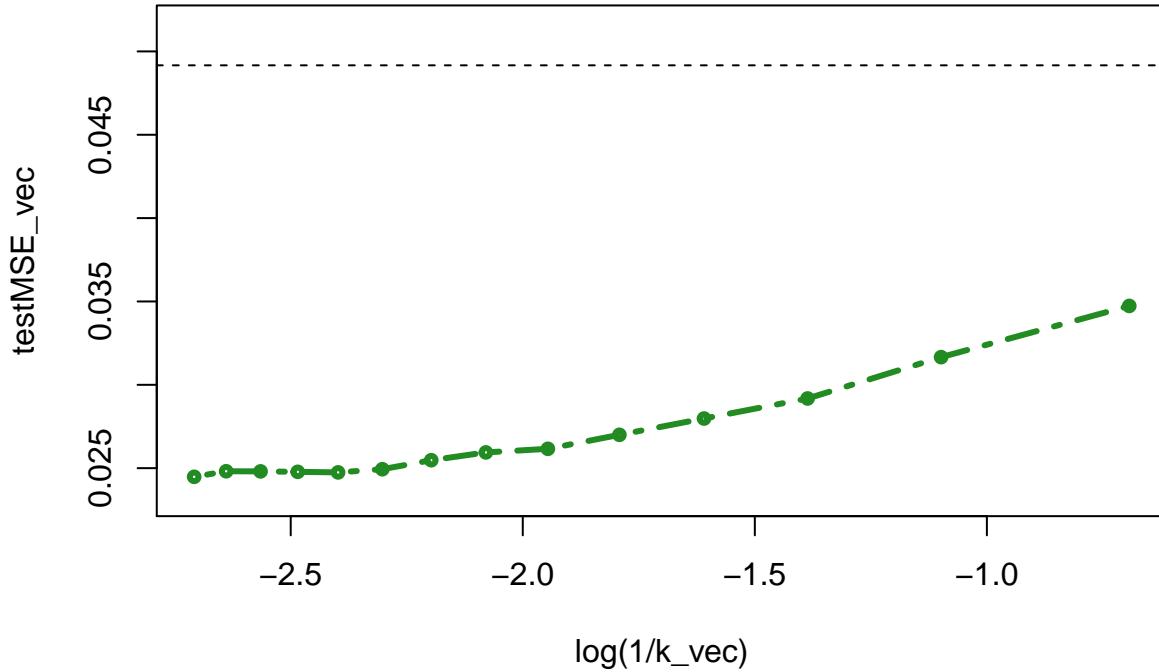


k-nn fit: k=12



```
plotMSE_nlinear <- PlotModelError(sim_nlinear, plot_ls_nlinear, plot_knn_nlinear)
```

Test set mean squared error



- Observation

When the true relationship is highly non-linear, k-NN gives much better result than linear regression, for all values of k . The test MSE of linear regression increases dramatically with the extent of non-linearity while that of k-NN changes little.

Generate data

```
Xtrain=matrix(runif(N_train*20, -1, 1), nrow=N_train)
Xtest=matrix(runif(N_test*20, -1, 1), nrow=N_test)
Ytrain=sin(2*Xtrain[,1])+2+rnorm(N_train)
Ytest=sin(2*Xtest[,1])+2+rnorm(N_test)

for (pp in 1:5) {
  par(mfrow = c(1, 4), mai=c(0.2,0.2,0.2,0.2))
  for (p in seq((pp-1)*4+1, (pp)*4)) {

    Xtrain=matrix(runif(N_train*p, -1, 1), nrow=N_train)
    Xtest=matrix(runif(N_test*p, -1, 1), nrow=N_test)
    Ytrain=sin(2*Xtrain[,1])+2+0.3*rnorm(N_train)
    Ytest=sin(2*Xtest[,1])+2+0.3*rnorm(N_test)

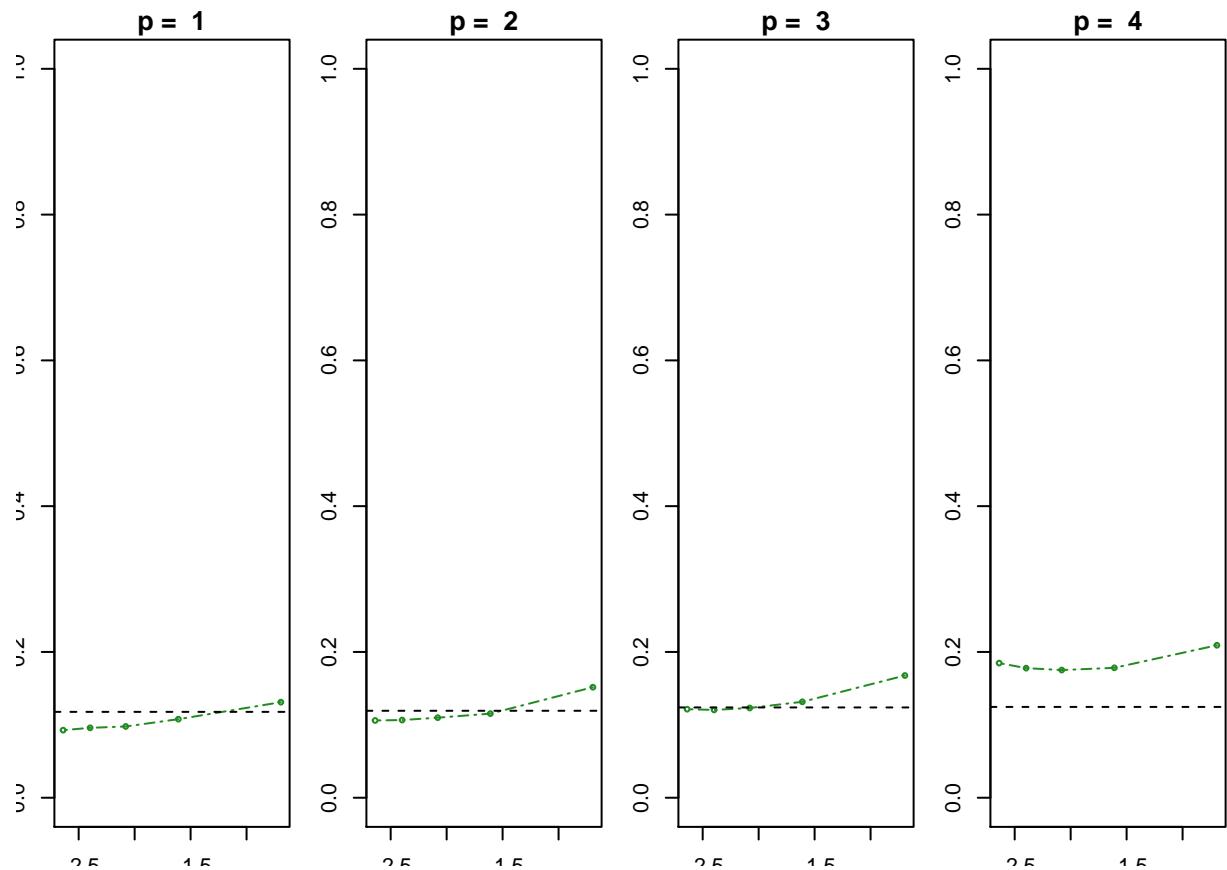
    train=data.table(y=Ytrain, Xtrain[,1:p])
    colnames(train)=c("y", paste0("x", 1:p))
    test=data.table(y=Ytest, Xtest[,1:p])
    colnames(test)=c("y", paste0("x", 1:p))
    linear=lm(y~., data=train)

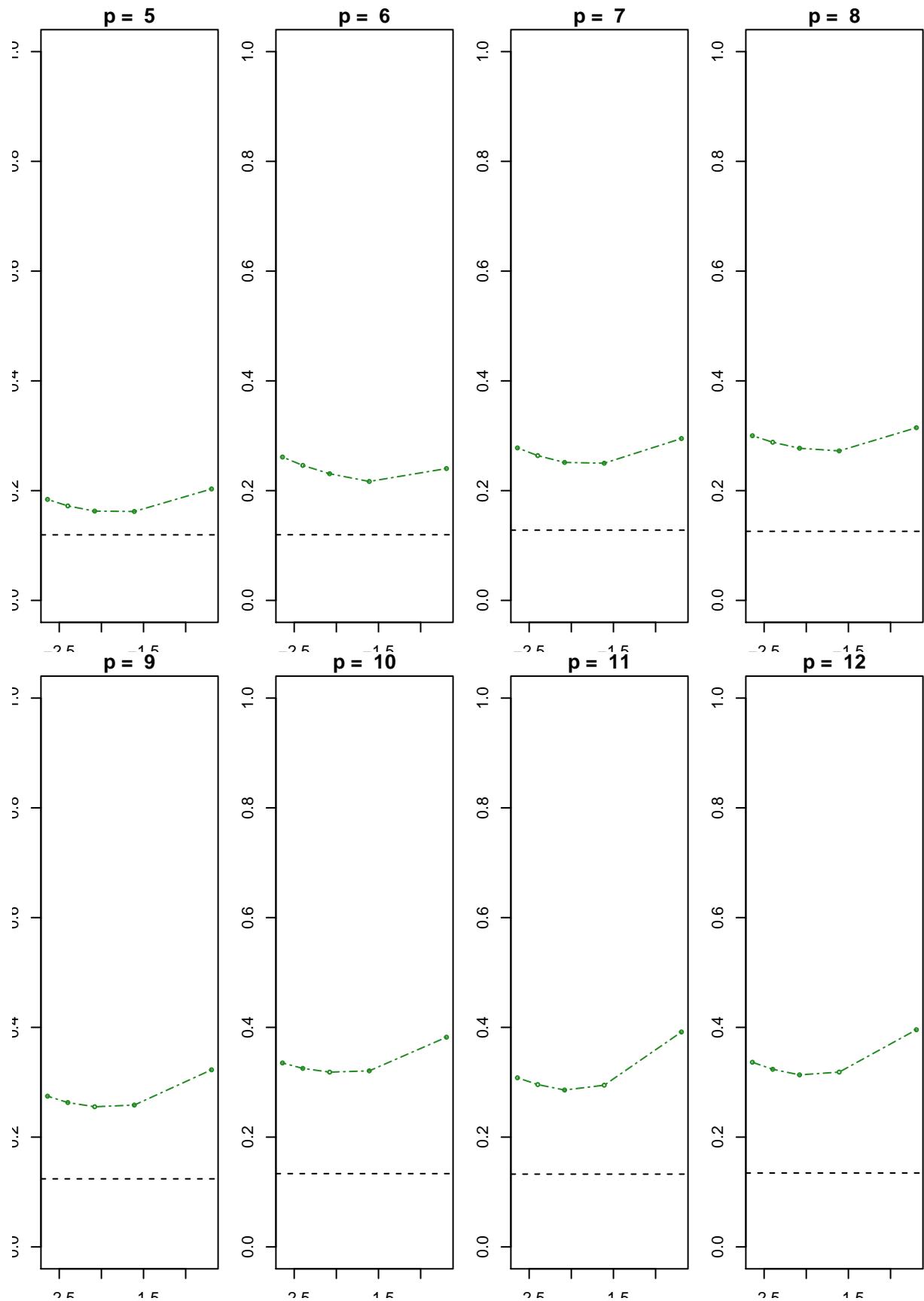
    lsMSE=mean((predict(linear, test)-test$y)^2)
```

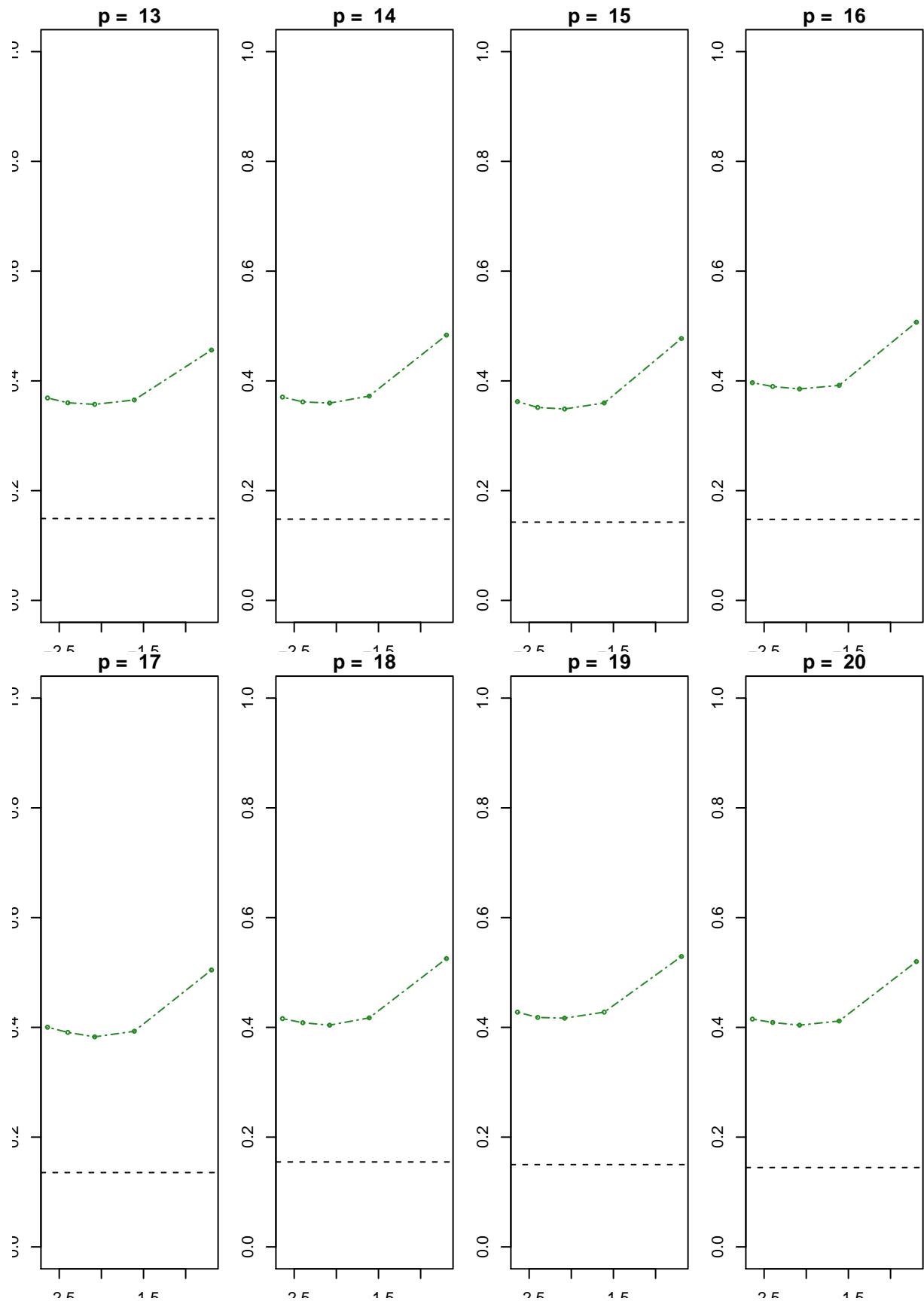
```

k_vec=seq(2, 15, 3)
testMSE_vec=numeric(length=length(k_vec))
for (i in 1:length(k_vec)){
  near=kknn(y~., train=train, test=test, k=k_vec[i], kernel='rectangular')
  testMSE_vec[i]=mean((near$fitted-test$y)^2)
}
plot(log(1/k_vec), testMSE_vec,
      ylim = c(0.,1.),
      type="n")
title(main=paste("p = ",p))
lines(log(1/k_vec), testMSE_vec, lty=6, lwd=1, col="forestgreen", type="o", cex=0.5 )
abline(h=lsMSE, col="black", lty=2)
}
}

```







- Observation

The increase in dimension only causes a small deterioration in the linear regression test set MSE, but it causes substantial increase in the MSE for k-NN. This is the curse-of-dimensionality problem, which results from the fact that in higher dimensions there is effectively a reduction in sample size. The general rule is linear regression will outperform k-NN method when there is a small number of observations per predictor, though in our simulation exercise this relationship does not hold monotonically.

Optional bonus question

- Our conclusion concerning the relative performance of linear regression and k-NN as the extent of model non-linearity varies will remain unchanged.
- We will have a wider range of values of k for which k-NN outperforms linear regression. In real world cases where the true relationship is barely linear, for any given p, an increase in the size of training sample boosts the number of sample points within each small neighborhood, which helps to improve the performance of k-NN. In other words, k-NN can now sustain more complex models without the peril of overfitting.
- Generally having a larger training set improves prediction accuracy in terms of test error by reducing the chance of overfitting.

Question 2

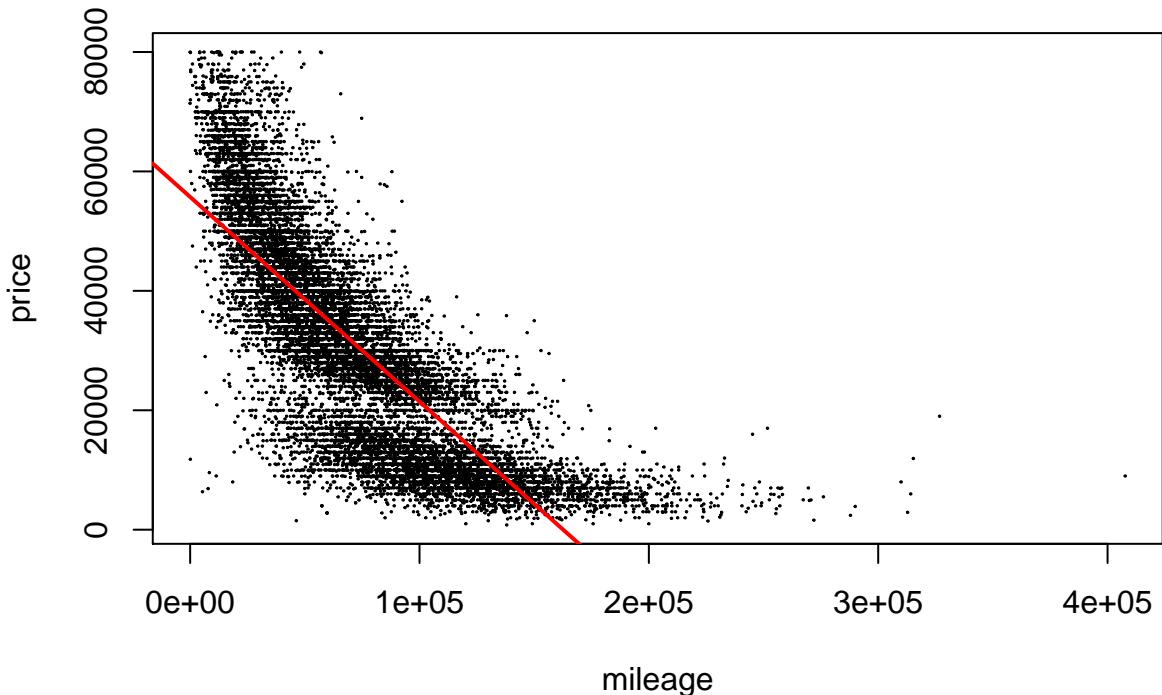
```
# set seed
set.seed(100)

# read data
download.file("https://github.com/ChicagoBoothML/MLClassData/raw/master/UsedCars/UsedCars.csv",
              "UsedCars.csv")
UsedCars <- read.csv(file="UsedCars.csv", head=TRUE, sep=", ")

# splitting data into training and validation
N = dim(UsedCars)[1]
train_index = sample(N, size = N * 0.75, replace = FALSE)
train = UsedCars[train_index, ]
test = UsedCars[-train_index, ]
```

Part 3

```
fit = lm(price ~ mileage, data = train)
plot(train$mileage, train$price, xlab = "mileage", ylab = "price", pch = 1, cex = 0.1)
abline(fit, col = "red", lwd = 2)
```



The linear regression can not capture the non-linear trend of the data.

Part 4

```

docv = function(x,y,set,nfold=10,doran=TRUE,verbose=TRUE,...)
{
  #a little error checking
  x = as.matrix(x)
  set = matrix(set, ncol = 1)
  if(!(is.matrix(x) | is.data.frame(x))) {cat('error in docv: x is not a matrix or data frame\n'); return(0)}
  if(!(is.vector(y))) {cat('error in docv: y is not a vector\n'); return(0)}
  if(!(length(y)==nrow(x))) {cat('error in docv: length(y) != nrow(x)\n'); return(0)}

  nset = nrow(set);
  n=length(y) #get dimensions

  if(n==nfold) doran=FALSE #no need to shuffle if you are doing them all.
  cat('in docv: nset,n,nfold: ',nset,n,nfold,'\n')
  lossv = rep(0,nset) #return values
  if(doran) {ii = sample(1:n,n); y=y[ii]; x=x[ii,,drop=FALSE]} #shuffle rows

  fs = round(n/nfold) # fold size
  for(i in 1:nfold) { #fold loop
    bot = (i-1)*fs+1;
    top = ifelse(i==nfold,n,i*fs);
    ii = bot:top
    if(verbose) cat('on fold: ',i,', range: ',bot,':',top,'\n')
    xin = x[-ii,,drop=FALSE];
    yin=y[-ii];
    xout=x[ii,,drop=FALSE];
  }
}

```

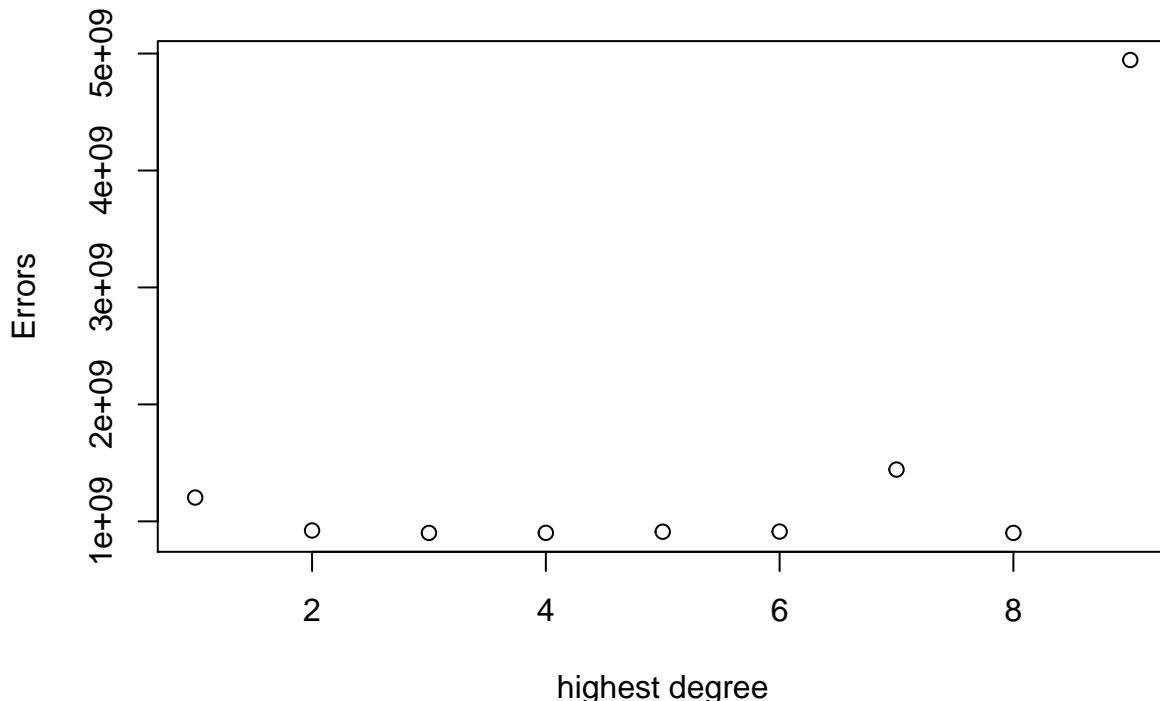
```

yout=y[ii]
xin = as.vector(xin)
xout = as.vector(xout)
datain = data.frame(x = xin, y = yin)
dataout = data.frame(x = xout, y = yout)
for(k in 1:nset)
{ #setting loop
  fit = lm(y ~ poly(x, set[k,]), data = datain)
  yhat = predict(fit, newdata = dataout)
  lossv[k]=lossv[k]+mean((yout-yhat)^2)
}
}
return(lossv)
}

#### run cv
D = c(1,2,3,4,5,6, 7, 8, 9) # potential degrees
errors = docv(train$mileage, train$price, D, nfold = 10)

## in docv: nset,n,nfold: 9 15047 10
## on fold: 1 , range: 1 : 1505
## on fold: 2 , range: 1506 : 3010
## on fold: 3 , range: 3011 : 4515
## on fold: 4 , range: 4516 : 6020
## on fold: 5 , range: 6021 : 7525
## on fold: 6 , range: 7526 : 9030
## on fold: 7 , range: 9031 : 10535
## on fold: 8 , range: 10536 : 12040
## on fold: 9 , range: 12041 : 13545
## on fold: 10 , range: 13546 : 15047
plot(D, errors, xlab = "highest degree", ylab = "Errors")

```



```

highest degree

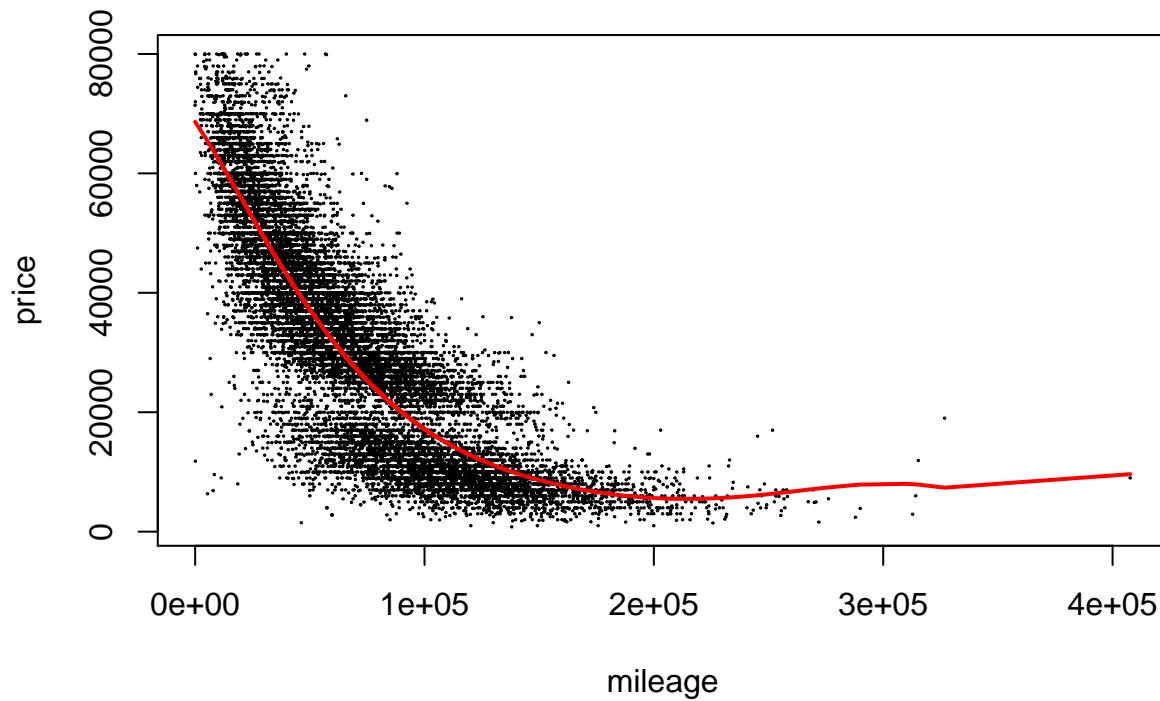
best.degree = D[which.min(errors)] # find the best degree

best.degree

## [1] 3

fit = lm(price ~ poly(mileage, 7), data = train)
plot(train$mileage, train$price, xlab = "mileage", ylab = "price", pch = 1, cex = 0.1)
fitted = predict(fit, newdata = data.frame(mileage = sort(train$mileage)))
lines(sort(train$mileage),fitted,col="red",lwd=2, cex.lab=2)

```



```
# error
sqrt(mean((test$price -predict(fit, test))^2))
```

```
## [1] 10537.12
```

Due to randomness of the code, the result might be different for each run.

Part 5

```
#####
## kNN
## Warning: super slow
#####

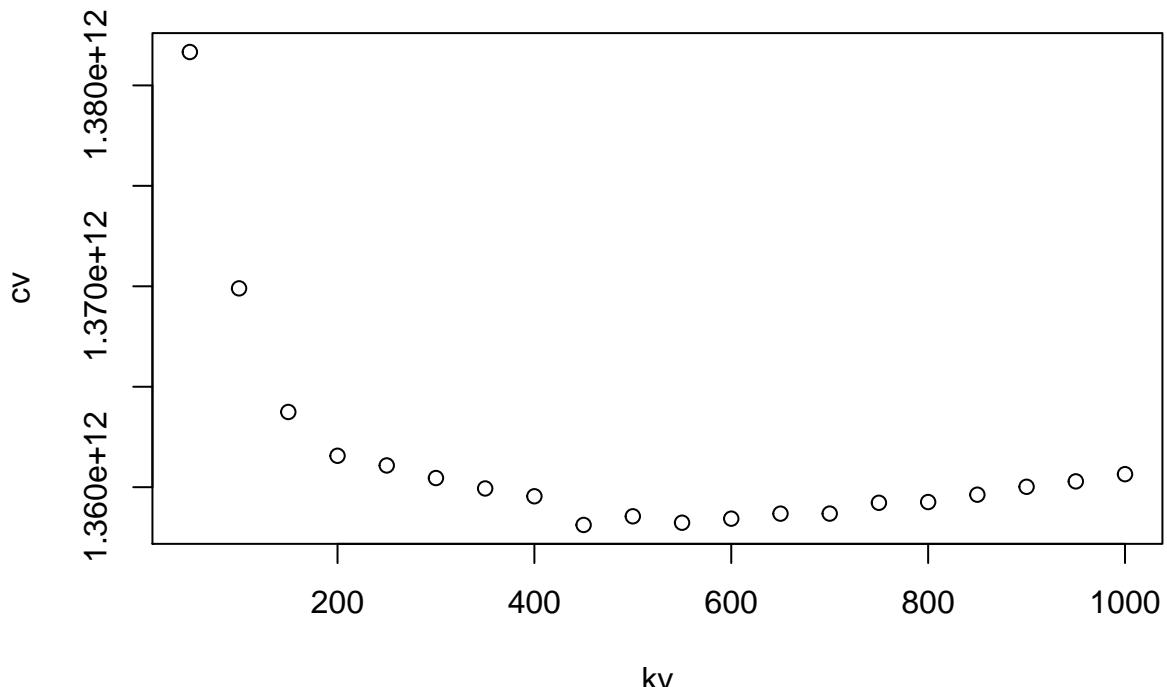
library(MASS)
library(knn)
download.file("https://raw.githubusercontent.com/ChicagoBoothML/HelpR/master/docv.R", "docv.R")
source("docv.R") #this has docvknn used below
set.seed(99) #always set the seed!

kv = 1:20 * 50 #these are the k values (k as in kNN) we will try
# since the possible k might be very large, we pick k every 50, from 50 to 1000

#does cross-validation for training data (x,y).

cv = docvknn(matrix(train$mileage,ncol=1),train$price,kv,nfold=10)

## in docv: nset,n,nfold: 20 15047 10
## on fold: 1 , range: 1 : 1505
## on fold: 2 , range: 1506 : 3010
## on fold: 3 , range: 3011 : 4515
## on fold: 4 , range: 4516 : 6020
## on fold: 5 , range: 6021 : 7525
## on fold: 6 , range: 7526 : 9030
## on fold: 7 , range: 9031 : 10535
## on fold: 8 , range: 10536 : 12040
## on fold: 9 , range: 12041 : 13545
## on fold: 10 , range: 13546 : 15047
plot(kv, cv)
```

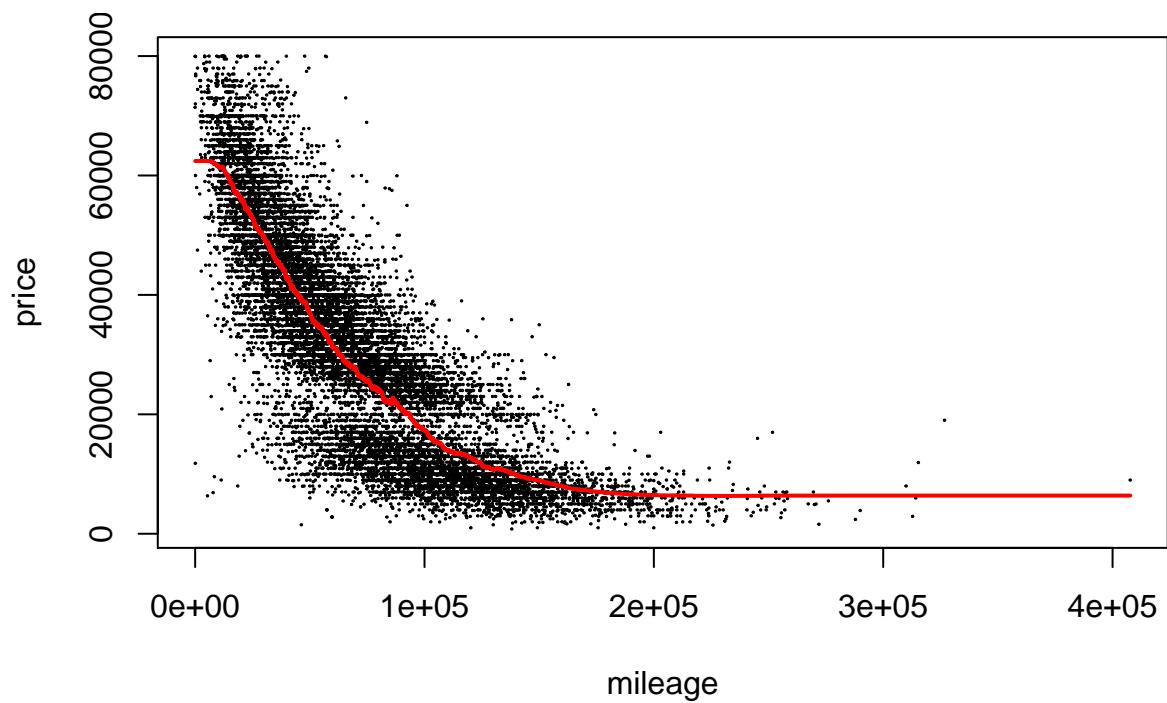


```

kvbest = kv[which.min(cv)]

kfbest = kknn(price~mileage,train,data.frame(mileage=sort(train$mileage)),k=kbest,kernel = "rectangular")
plot(train$mileage, train$price, xlab = "mileage", ylab = "price", pch = 1, cex = 0.1)
lines(sort(train$mileage),kfbest$fitted,col="red",lwd=2, cex.lab=2)

```



```

# error
kfbest = kknn(price~mileage,train,test,k=kbest,kernel = "rectangular")
sqrt(mean((test$price - kfbest$fitted.values)^2))

```

```

## [1] 9471.33
#####
## Tree
#####

library(rpart)      # package for trees
library(rpart.plot) # package that enhances plotting capabilities for rpart
library(MASS)      # contains boston housing data

big.tree = rpart(price~mileage, data=train,
                 control=rpart.control(minsplit=5, cp=0.0001, xval=10))

nbig = length(unique(big.tree$where))
cat('size of big tree: ', nbig, '\n')

## size of big tree: 158
(cptable = printcp(big.tree))

##
## Regression tree:
## rpart(formula = price ~ mileage, data = train, control = rpart.control(minsplit = 5,
##           cp = 1e-04, xval = 10))
##
## Variables actually used in tree construction:
## [1] mileage
##
## Root node error: 5.008e+12/15047 = 332822195
##
## n= 15047
##
##          CP nsplit rel error  xerror      xstd
## 1  0.53908774      0  1.00000 1.00012 0.0091403
## 2  0.07836283      1  0.46091 0.46211 0.0052704
## 3  0.06683151      2  0.38255 0.38425 0.0050372
## 4  0.01045803      3  0.31572 0.31788 0.0043603
## 5  0.00912934      4  0.30526 0.30777 0.0042855
## 6  0.00766216      5  0.29613 0.29937 0.0042399
## 7  0.00738518      6  0.28847 0.29168 0.0041789
## 8  0.00178057      7  0.28108 0.28469 0.0041277
## 9  0.00157085      8  0.27930 0.28273 0.0041088
## 10 0.00134710      9  0.27773 0.28156 0.0041206
## 11 0.00132704     10 0.27638 0.28052 0.0041345
## 12 0.00126318     11 0.27506 0.27973 0.0041277
## 13 0.00110278     12 0.27379 0.27849 0.0041112
## 14 0.00103506     13 0.27269 0.27704 0.0040984
## 15 0.00091311     14 0.27166 0.27612 0.0040947
## 16 0.00043113     15 0.27074 0.27475 0.0040803
## 17 0.00040747     16 0.27031 0.27520 0.0041377
## 18 0.00029303     17 0.26990 0.27489 0.0041368
## 19 0.00028726     18 0.26961 0.27490 0.0041626
## 20 0.00025328     21 0.26875 0.27486 0.0041614
## 21 0.00025269     23 0.26824 0.27508 0.0041753

```

```

## 22 0.00022213      24  0.26799 0.27489 0.0041764
## 23 0.00020287      25  0.26777 0.27495 0.0041802
## 24 0.00018330      26  0.26757 0.27651 0.0042260
## 25 0.00017980      27  0.26738 0.27692 0.0042314
## 26 0.00016158      28  0.26720 0.27726 0.0042383
## 27 0.00015544      29  0.26704 0.27893 0.0042699
## 28 0.00014731      35  0.26611 0.28012 0.0043119
## 29 0.00014230      44  0.26478 0.28121 0.0043374
## 30 0.00014075      46  0.26450 0.28224 0.0043587
## 31 0.00014058      48  0.26422 0.28255 0.0043676
## 32 0.00013823      49  0.26408 0.28261 0.0043695
## 33 0.00013642      57  0.26280 0.28309 0.0043741
## 34 0.00013007      66  0.26158 0.28430 0.0043885
## 35 0.00012671      74  0.26054 0.28505 0.0043973
## 36 0.00012496      75  0.26041 0.28611 0.0044358
## 37 0.00012488      82  0.25950 0.28626 0.0044364
## 38 0.00012329      84  0.25925 0.28628 0.0044364
## 39 0.00012118      85  0.25912 0.28652 0.0044388
## 40 0.00012094      86  0.25900 0.28680 0.0044382
## 41 0.00011797      89  0.25860 0.28718 0.0044439
## 42 0.00011733      90  0.25848 0.28770 0.0044587
## 43 0.00011518     101  0.25715 0.28818 0.0044698
## 44 0.00011452     108  0.25635 0.28859 0.0044759
## 45 0.00011401     109  0.25623 0.28863 0.0044749
## 46 0.00011353     110  0.25612 0.28868 0.0044769
## 47 0.00011290     111  0.25600 0.28873 0.0044682
## 48 0.00011170     113  0.25578 0.28874 0.0044676
## 49 0.00011024     118  0.25520 0.28909 0.0044589
## 50 0.00011006     119  0.25509 0.28933 0.0044742
## 51 0.00010932     120  0.25498 0.28941 0.0044714
## 52 0.00010537     132  0.25358 0.29131 0.0044829
## 53 0.00010398     136  0.25314 0.29348 0.0045150
## 54 0.00010371     137  0.25303 0.29362 0.0045164
## 55 0.00010248     149  0.25171 0.29408 0.0045252
## 56 0.00010151     156  0.25095 0.29446 0.0045291
## 57 0.00010000     157  0.25085 0.29672 0.0045885

##          CP nsplit rel error      xerror      xstd
## 1  0.5390877357      0 1.0000000 1.0001207 0.009140319
## 2  0.0783628292      1 0.4609123 0.4621082 0.005270426
## 3  0.0668315064      2 0.3825494 0.3842476 0.005037162
## 4  0.0104580334      3 0.3157179 0.3178814 0.004360294
## 5  0.0091293448      4 0.3052599 0.3077734 0.004285485
## 6  0.0076621629      5 0.2961306 0.2993714 0.004239868
## 7  0.0073851837      6 0.2884684 0.2916778 0.004178879
## 8  0.0017805692      7 0.2810832 0.2846904 0.004127741
## 9  0.0015708518      8 0.2793026 0.2827267 0.004108769
## 10 0.0013470994     9 0.2777318 0.2815625 0.004120566
## 11 0.0013270404    10 0.2763847 0.2805225 0.004134491
## 12 0.0012631817    11 0.2750576 0.2797316 0.004127658
## 13 0.0011027784    12 0.2737945 0.2784881 0.004111244
## 14 0.0010350642    13 0.2726917 0.2770408 0.004098444
## 15 0.0009131128    14 0.2716566 0.2761213 0.004094681
## 16 0.0004311265    15 0.2707435 0.2747516 0.004080315

```

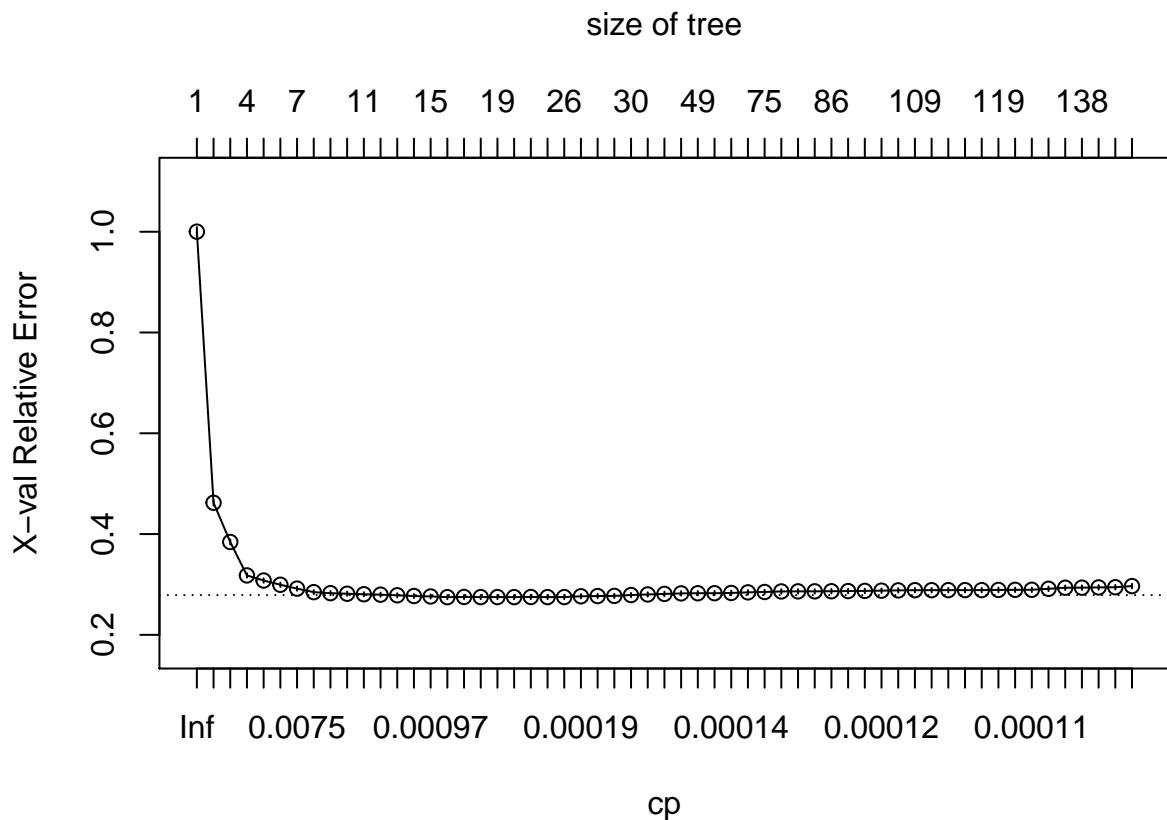
```

## 17 0.0004074681      16 0.2703124 0.2752013 0.004137656
## 18 0.0002930337      17 0.2699049 0.2748866 0.004136775
## 19 0.0002872586      18 0.2696119 0.2748980 0.004162551
## 20 0.0002532837      21 0.2687501 0.2748595 0.004161396
## 21 0.0002526870      23 0.2682435 0.2750782 0.004175347
## 22 0.0002221260      24 0.2679908 0.2748920 0.004176384
## 23 0.0002028733      25 0.2677687 0.2749462 0.004180235
## 24 0.0001832985      26 0.2675658 0.2765097 0.004225985
## 25 0.0001798018      27 0.2673825 0.2769152 0.004231387
## 26 0.0001615783      28 0.2672027 0.2772587 0.004238251
## 27 0.0001554357      29 0.2670412 0.2789293 0.004269940
## 28 0.0001473055      35 0.2661086 0.2801205 0.004311867
## 29 0.0001422984      44 0.2647824 0.2812088 0.004337355
## 30 0.0001407465      46 0.2644978 0.2822362 0.004358691
## 31 0.0001405768      48 0.2642163 0.2825452 0.004367595
## 32 0.0001382300      49 0.2640758 0.2826120 0.004369508
## 33 0.0001364228      57 0.2628049 0.2830919 0.004374056
## 34 0.0001300727      66 0.2615771 0.2843005 0.004388472
## 35 0.0001267091      74 0.2605365 0.2850488 0.004397319
## 36 0.0001249585      75 0.2604098 0.2861134 0.004435758
## 37 0.0001248838      82 0.2594980 0.2862623 0.004436365
## 38 0.0001232930      84 0.2592482 0.2862825 0.004436424
## 39 0.0001211767      85 0.2591249 0.2865154 0.004438816
## 40 0.0001209394      86 0.2590037 0.2868041 0.004438190
## 41 0.0001179662      89 0.2585956 0.2871786 0.004443877
## 42 0.0001173263      90 0.2584777 0.2876969 0.004458678
## 43 0.0001151809      101 0.2571519 0.2881816 0.004469789
## 44 0.0001145172      108 0.2563457 0.2885877 0.004475851
## 45 0.0001140117      109 0.2562312 0.2886310 0.004474892
## 46 0.0001135314      110 0.2561171 0.2886762 0.004476863
## 47 0.0001129039      111 0.2560036 0.2887261 0.004468185
## 48 0.0001116955      113 0.2557778 0.2887443 0.004467568
## 49 0.0001102378      118 0.2552003 0.2890862 0.004458926
## 50 0.0001100552      119 0.2550900 0.2893328 0.004474158
## 51 0.0001093245      120 0.2549800 0.2894150 0.004471449
## 52 0.0001053704      132 0.2535835 0.2913079 0.004482856
## 53 0.0001039798      136 0.2531352 0.2934839 0.004515004
## 54 0.0001037116      137 0.2530312 0.2936236 0.004516425
## 55 0.0001024819      149 0.2517074 0.2940842 0.004525157
## 56 0.0001015093      156 0.2509533 0.2944600 0.004529065
## 57 0.0001000000      157 0.2508518 0.2967185 0.004588457

(bestcp = cptable[ which.min(cptable[, "xerror"])), "CP" ])    # this is the optimal cp parameter

## [1] 0.0004311265
plotcp(big.tree) # plot results

```

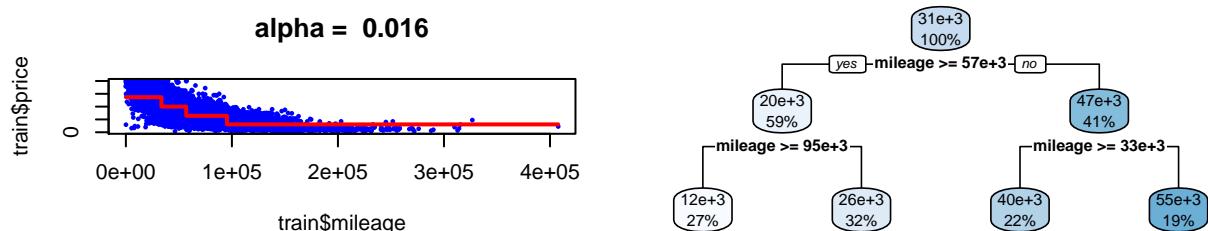
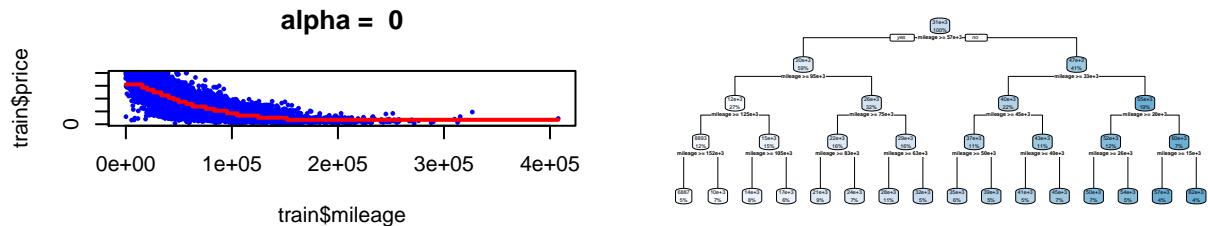
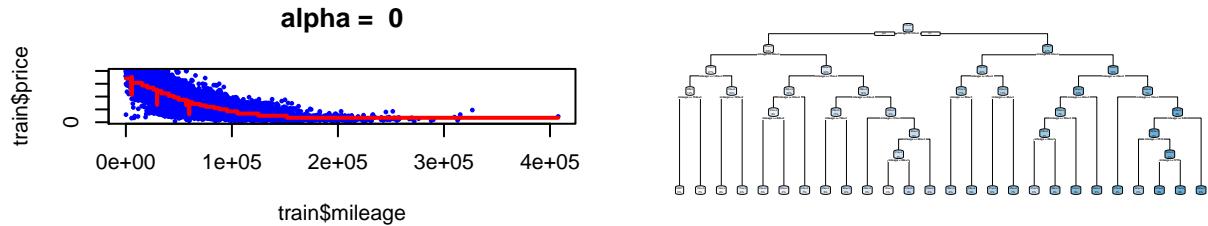


```

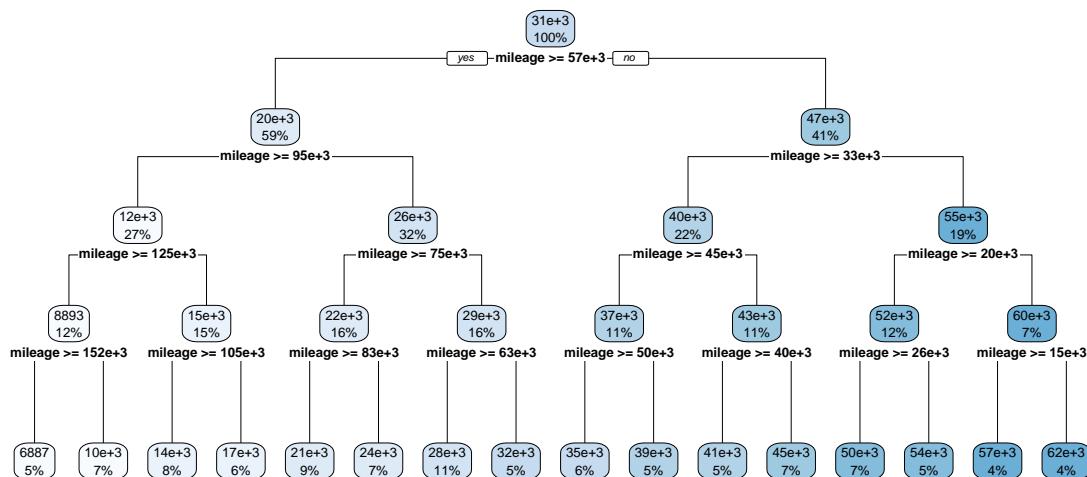
# show fit from some trees
oo = order(train$mileage)
cpvec = c(bestcp / 2, bestcp,.0157)

par(mfrow=c(3,2))
for(i in 1:3) {
  plot(train$mileage,train$price,pch=16,col='blue',cex=.5)
  ptree = prune(big.tree,cp=cpvec[i])
  pfit = predict(ptree)
  lines(train$mileage[oo],pfit[oo],col='red',lwd=2)
  title(paste('alpha = ',round(cpvec[i],3)))
  rpart.plot(ptree)
}

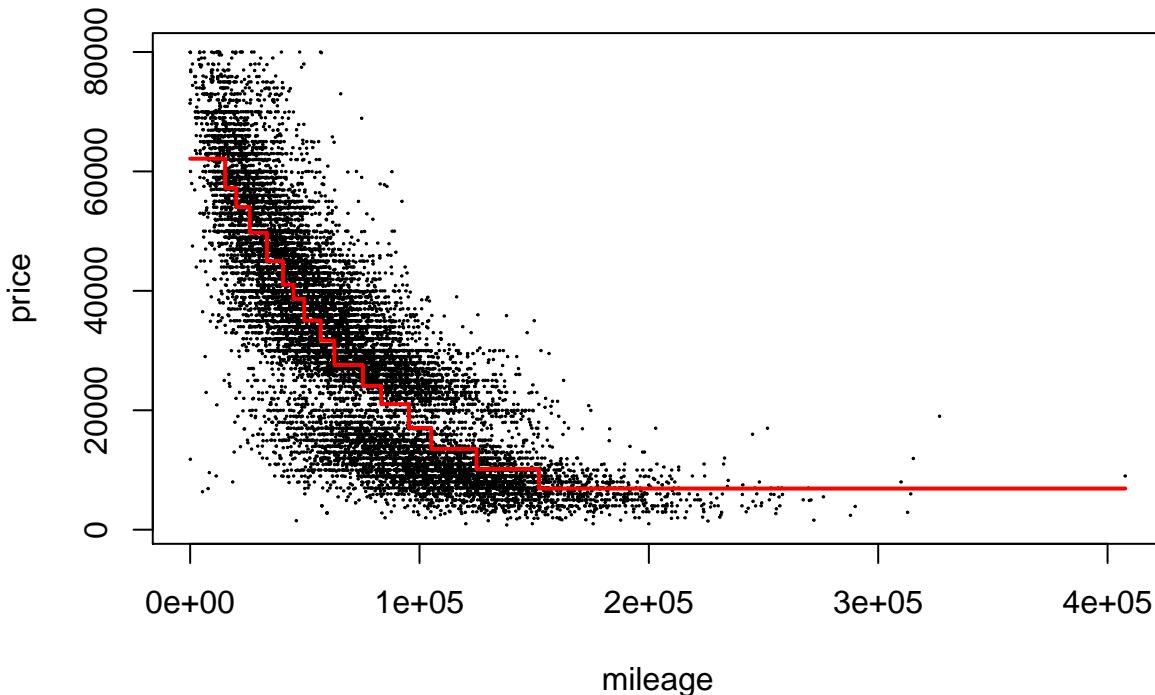
```



```
par(mfrow=c(1,1))
best.tree = prune(big.tree, cp=bestcp)
rpart.plot(best.tree)
```



```
plot(train$mileage, train$price, xlab = "mileage", ylab = "price", pch = 1, cex = 0.1)
lines(sort(train$mileage), predict(best.tree) [oo], col="red", lwd=2, cex.lab=2)
```



```
# error
sqrt(mean((test$price - predict(best.tree, test))^2))
```

```
## [1] 9545.132
```

From figures above, I would choose kNN because it's robust and smooth. Tree and high-order polynomials are less smooth and very sensitive to extreme values.

Part 6

```
#####
# 6
# Add one more variable

#####
## kNN
## Warning: super slow
#####

library(MASS)
library(kknn)
set.seed(99) #always set the seed!

kv = 1:20 * 10

cv = docvknn(scale(as.matrix(cbind(train$mileage, train$year))),train$price,kv,nfold=10)

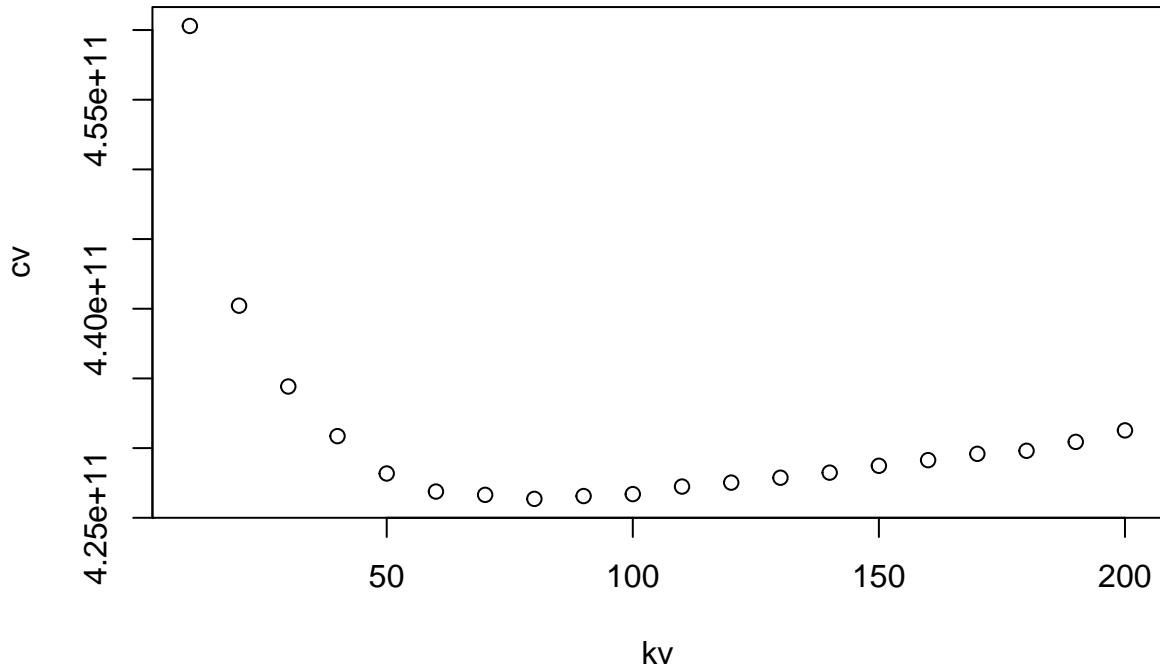
## in docv: nset,n,nfold: 20 15047 10
## on fold: 1 , range: 1 : 1505
```

```

## on fold: 2 , range: 1506 : 3010
## on fold: 3 , range: 3011 : 4515
## on fold: 4 , range: 4516 : 6020
## on fold: 5 , range: 6021 : 7525
## on fold: 6 , range: 7526 : 9030
## on fold: 7 , range: 9031 : 10535
## on fold: 8 , range: 10536 : 12040
## on fold: 9 , range: 12041 : 13545
## on fold: 10 , range: 13546 : 15047

plot(kv, cv)

```



```

kbest = kv[which.min(cv)]

train.scaled = train
train.scaled$year = scale(train.scaled$year)
train.scaled$mileage = scale(train.scaled$mileage)
test.scaled = test
test.scaled$year = scale(test.scaled$year)
test.scaled$mileage = scale(test.scaled$mileage)

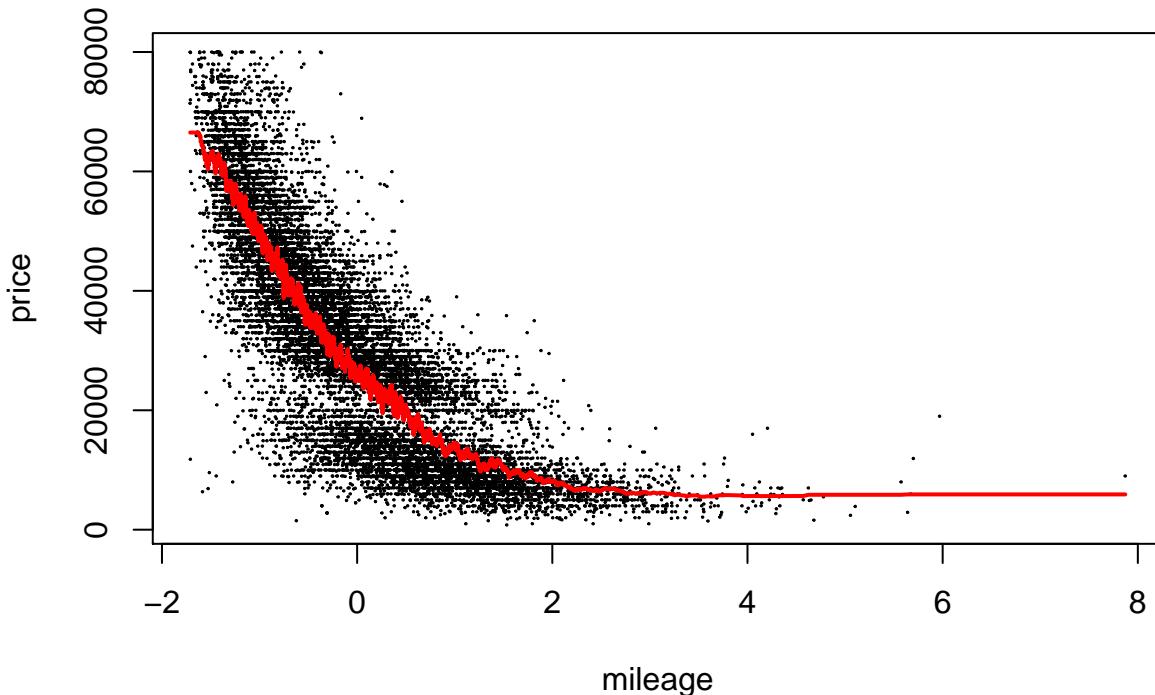
kfbest = kknn(price~mileage+price,train.scaled,data.frame(mileage=sort(train.scaled$mileage)),k=kbest,k=1)

## Warning in model.matrix.default(mt, mf): the response appeared on the
## right-hand side and was dropped

## Warning in model.matrix.default(mt, mf): problem with term 2 in
## model.matrix: no columns are assigned

plot(train.scaled$mileage, train$price, xlab = "mileage", ylab = "price", pch = 1, cex = 0.1)
lines(sort(train.scaled$mileage),kfbest$fitted,col="red",lwd=2, cex.lab=2)

```



```

# error
kfbest = kknn(price~mileage,train.scaled,test.scaled,k=kbest,kernel = "rectangular")
sqrt(mean((test.scaled$price - kfbest$fitted.values)^2))

## [1] 9547.901
#####
## Tree
#####
library(rpart)      # package for trees
library(rpart.plot) # package that enhances plotting capabilities for rpart
library(MASS)      # contains boston housing data

big.tree = rpart(price~mileage+price, data=train,
                 control=rpart.control(minsplit=5,cp=0.0001,xval=10))

## Warning in model.matrix.default(attr(frame, "terms"), frame): the response
## appeared on the right-hand side and was dropped
## Warning in model.matrix.default(attr(frame, "terms"), frame): problem with
## term 2 in model.matrix: no columns are assigned
nbig = length(unique(big.tree$where))
cat('size of big tree: ',nbig,'\n')

## size of big tree: 158
(cptable = printcp(big.tree))

##
## Regression tree:
## rpart(formula = price ~ mileage + price, data = train, control = rpart.control(minsplit = 5,
##         cp = 1e-04, xval = 10))
##

```

```

## Variables actually used in tree construction:
## [1] mileage
##
## Root node error: 5.008e+12/15047 = 332822195
##
## n= 15047
##
##          CP nsplit rel error  xerror      xstd
## 1  0.53908774      0  1.00000 1.00012 0.0091403
## 2  0.07836283      1  0.46091 0.46211 0.0052704
## 3  0.06683151      2  0.38255 0.38425 0.0050372
## 4  0.01045803      3  0.31572 0.31788 0.0043603
## 5  0.00912934      4  0.30526 0.30777 0.0042855
## 6  0.00766216      5  0.29613 0.29937 0.0042399
## 7  0.00738518      6  0.28847 0.29168 0.0041789
## 8  0.00178057      7  0.28108 0.28469 0.0041277
## 9  0.00157085      8  0.27930 0.28273 0.0041088
## 10 0.00134710      9  0.27773 0.28156 0.0041206
## 11 0.00132704     10 0.27638 0.28052 0.0041345
## 12 0.00126318     11 0.27506 0.27973 0.0041277
## 13 0.00110278     12 0.27379 0.27849 0.0041112
## 14 0.00103506     13 0.27269 0.27704 0.0040984
## 15 0.00091311     14 0.27166 0.27612 0.0040947
## 16 0.00043113     15 0.27074 0.27475 0.0040803
## 17 0.00040747     16 0.27031 0.27520 0.0041377
## 18 0.00029303     17 0.26990 0.27489 0.0041368
## 19 0.00028726     18 0.26961 0.27490 0.0041626
## 20 0.00025328     21 0.26875 0.27486 0.0041614
## 21 0.00025269     23 0.26824 0.27508 0.0041753
## 22 0.00022213     24 0.26799 0.27489 0.0041764
## 23 0.00020287     25 0.26777 0.27495 0.0041802
## 24 0.00018330     26 0.26757 0.27651 0.0042260
## 25 0.00017980     27 0.26738 0.27692 0.0042314
## 26 0.00016158     28 0.26720 0.27726 0.0042383
## 27 0.00015544     29 0.26704 0.27893 0.0042699
## 28 0.00014731     35 0.26611 0.28012 0.0043119
## 29 0.00014230     44 0.26478 0.28121 0.0043374
## 30 0.00014075     46 0.26450 0.28224 0.0043587
## 31 0.00014058     48 0.26422 0.28255 0.0043676
## 32 0.00013823     49 0.26408 0.28261 0.0043695
## 33 0.00013642     57 0.26280 0.28309 0.0043741
## 34 0.00013007     66 0.26158 0.28430 0.0043885
## 35 0.00012671     74 0.26054 0.28505 0.0043973
## 36 0.00012496     75 0.26041 0.28611 0.0044358
## 37 0.00012488     82 0.25950 0.28626 0.0044364
## 38 0.00012329     84 0.25925 0.28628 0.0044364
## 39 0.00012118     85 0.25912 0.28652 0.0044388
## 40 0.00012094     86 0.25900 0.28680 0.0044382
## 41 0.00011797     89 0.25860 0.28718 0.0044439
## 42 0.00011733     90 0.25848 0.28770 0.0044587
## 43 0.00011518    101 0.25715 0.28818 0.0044698
## 44 0.00011452    108 0.25635 0.28859 0.0044759
## 45 0.00011401    109 0.25623 0.28863 0.0044749
## 46 0.00011353    110 0.25612 0.28868 0.0044769

```

```

## 47 0.00011290    111  0.25600 0.28873 0.0044682
## 48 0.00011170    113  0.25578 0.28874 0.0044676
## 49 0.00011024    118  0.25520 0.28909 0.0044589
## 50 0.00011006    119  0.25509 0.28933 0.0044742
## 51 0.00010932    120  0.25498 0.28941 0.0044714
## 52 0.00010537    132  0.25358 0.29131 0.0044829
## 53 0.00010398    136  0.25314 0.29348 0.0045150
## 54 0.00010371    137  0.25303 0.29362 0.0045164
## 55 0.00010248    149  0.25171 0.29408 0.0045252
## 56 0.00010151    156  0.25095 0.29446 0.0045291
## 57 0.00010000    157  0.25085 0.29672 0.0045885

##          CP nsplit rel error      xerror      xstd
## 1  0.5390877357      0 1.0000000 1.0001207 0.009140319
## 2  0.0783628292      1 0.4609123 0.4621082 0.005270426
## 3  0.0668315064      2 0.3825494 0.3842476 0.005037162
## 4  0.0104580334      3 0.3157179 0.3178814 0.004360294
## 5  0.0091293448      4 0.3052599 0.3077734 0.004285485
## 6  0.0076621629      5 0.2961306 0.2993714 0.004239868
## 7  0.0073851837      6 0.2884684 0.2916778 0.004178879
## 8  0.0017805692      7 0.2810832 0.2846904 0.004127741
## 9  0.0015708518      8 0.2793026 0.2827267 0.004108769
## 10 0.0013470994     9 0.2777318 0.2815625 0.004120566
## 11 0.0013270404    10 0.2763847 0.2805225 0.004134491
## 12 0.0012631817    11 0.2750576 0.2797316 0.004127658
## 13 0.0011027784    12 0.2737945 0.2784881 0.004111244
## 14 0.0010350642    13 0.2726917 0.2770408 0.004098444
## 15 0.0009131128    14 0.2716566 0.2761213 0.004094681
## 16 0.0004311265    15 0.2707435 0.2747516 0.004080315
## 17 0.0004074681    16 0.2703124 0.2752013 0.004137656
## 18 0.0002930337    17 0.2699049 0.2748866 0.004136775
## 19 0.0002872586    18 0.2696119 0.2748980 0.004162551
## 20 0.0002532837    21 0.2687501 0.2748595 0.004161396
## 21 0.0002526870    23 0.2682435 0.2750782 0.004175347
## 22 0.0002221260    24 0.2679908 0.2748920 0.004176384
## 23 0.0002028733    25 0.2677687 0.2749462 0.004180235
## 24 0.0001832985    26 0.2675658 0.2765097 0.004225985
## 25 0.0001798018    27 0.2673825 0.2769152 0.004231387
## 26 0.0001615783    28 0.2672027 0.2772587 0.004238251
## 27 0.0001554357    29 0.2670412 0.2789293 0.004269940
## 28 0.0001473055    35 0.2661086 0.2801205 0.004311867
## 29 0.0001422984    44 0.2647824 0.2812088 0.004337355
## 30 0.0001407465    46 0.2644978 0.2822362 0.004358691
## 31 0.0001405768    48 0.2642163 0.2825452 0.004367595
## 32 0.0001382300    49 0.2640758 0.2826120 0.004369508
## 33 0.0001364228    57 0.2628049 0.2830919 0.004374056
## 34 0.0001300727    66 0.2615771 0.2843005 0.004388472
## 35 0.0001267091    74 0.2605365 0.2850488 0.004397319
## 36 0.0001249585    75 0.2604098 0.2861134 0.004435758
## 37 0.0001248838    82 0.2594980 0.2862623 0.004436365
## 38 0.0001232930    84 0.2592482 0.2862825 0.004436424
## 39 0.0001211767    85 0.2591249 0.2865154 0.004438816
## 40 0.0001209394    86 0.2590037 0.2868041 0.004438190
## 41 0.0001179662    89 0.2585956 0.2871786 0.004443877

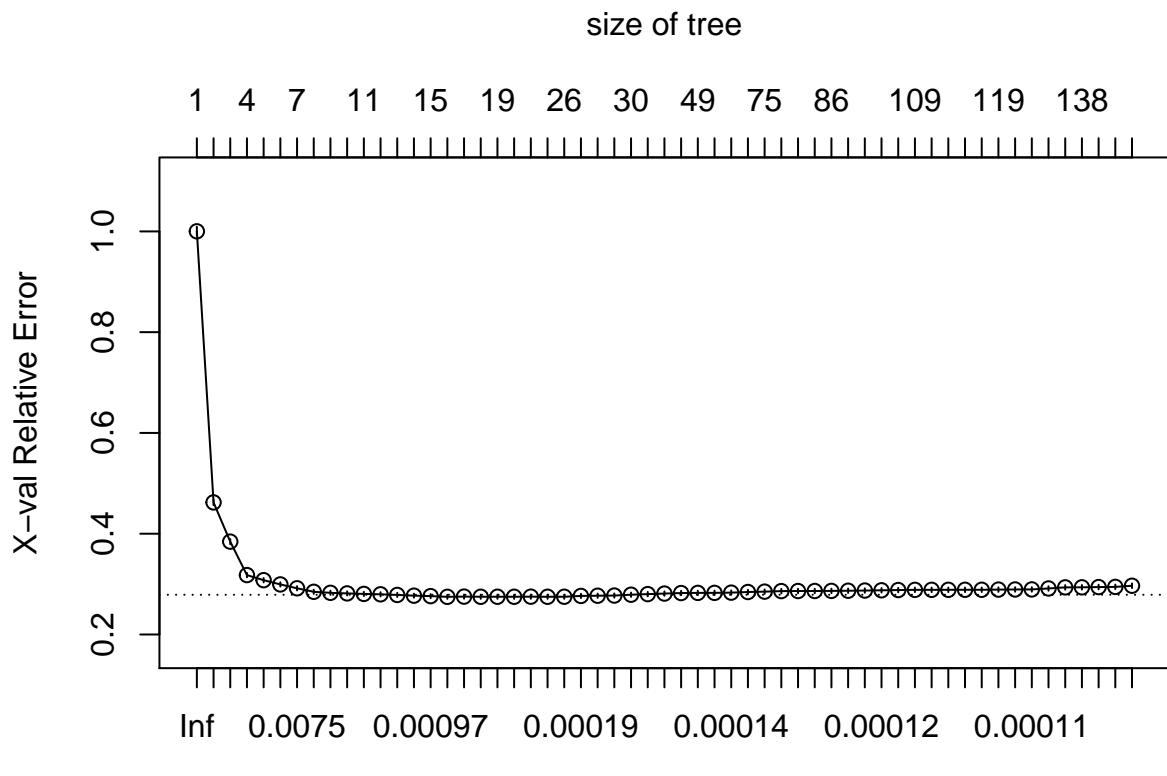
```

```

## 42 0.0001173263      90 0.2584777 0.2876969 0.004458678
## 43 0.0001151809     101 0.2571519 0.2881816 0.004469789
## 44 0.0001145172     108 0.2563457 0.2885877 0.004475851
## 45 0.0001140117     109 0.2562312 0.2886310 0.004474892
## 46 0.0001135314     110 0.2561171 0.2886762 0.004476863
## 47 0.0001129039     111 0.2560036 0.2887261 0.004468185
## 48 0.0001116955     113 0.2557778 0.2887443 0.004467568
## 49 0.0001102378     118 0.2552003 0.2890862 0.004458926
## 50 0.0001100552     119 0.2550900 0.2893328 0.004474158
## 51 0.0001093245     120 0.2549800 0.2894150 0.004471449
## 52 0.0001053704     132 0.2535835 0.2913079 0.004482856
## 53 0.0001039798     136 0.2531352 0.2934839 0.004515004
## 54 0.0001037116     137 0.2530312 0.2936236 0.004516425
## 55 0.0001024819     149 0.2517074 0.2940842 0.004525157
## 56 0.0001015093     156 0.2509533 0.2944600 0.004529065
## 57 0.0001000000     157 0.2508518 0.2967185 0.004588457

(bestcp = cptable[ which.min(cptable[, "xerror"]), "CP" ])    # this is the optimal cp parameter
## [1] 0.0004311265
plotcp(big.tree) # plot results

```



```

# show fit from some trees
oo = order(train$mileage)
cpvec = c(bestcp / 2, bestcp,.0157)

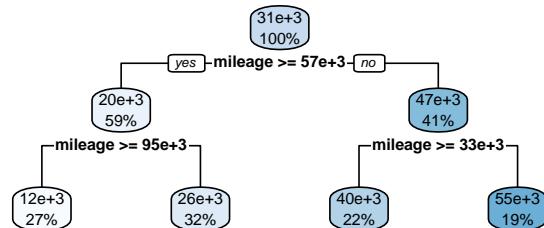
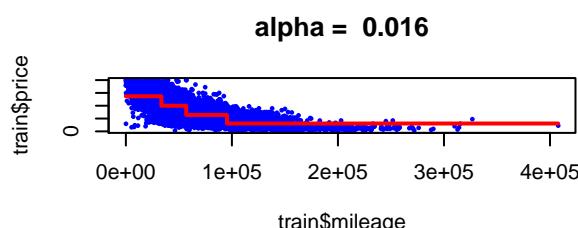
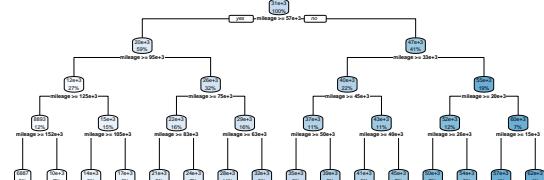
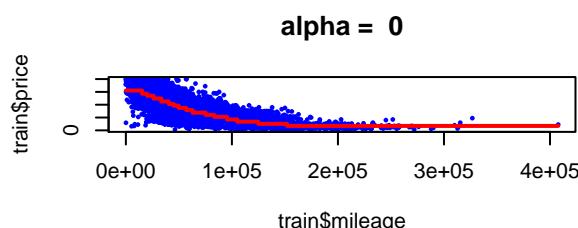
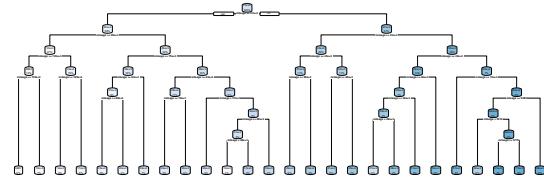
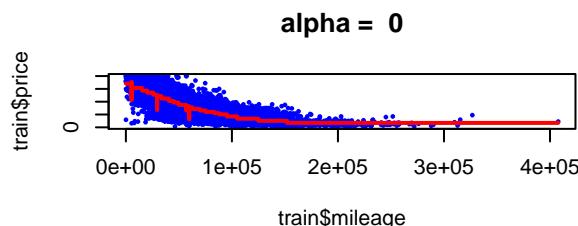
par(mfrow=c(3,2))
for(i in 1:3) {

```

```

plot(train$mileage,train$price,pch=16,col='blue',cex=.5)
ptree = prune(big.tree,cp=cpvec[i])
pfit = predict(ptree)
lines(train$mileage[oo],pfit[oo],col='red',lwd=2)
title(paste('alpha = ',round(cpvec[i],3)))
rpart.plot(ptree)
}

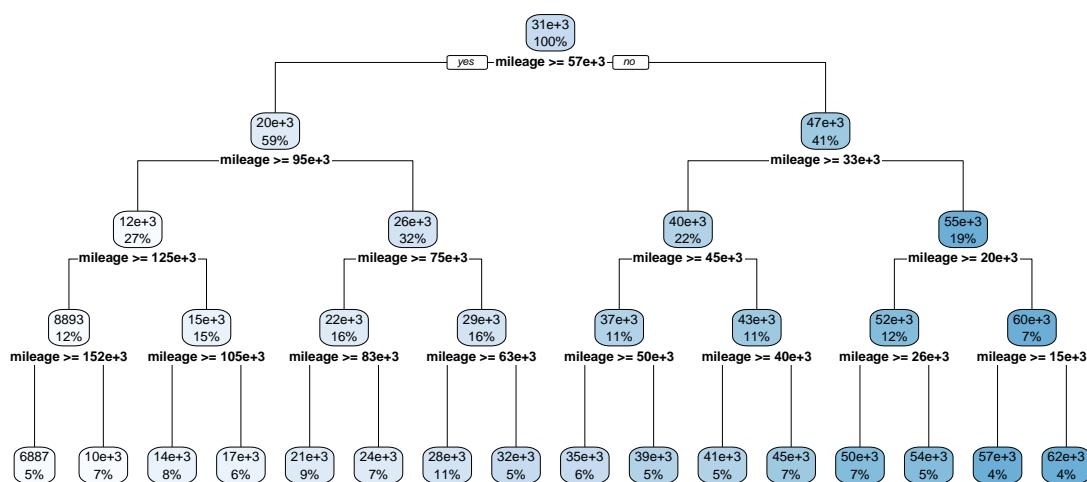
```



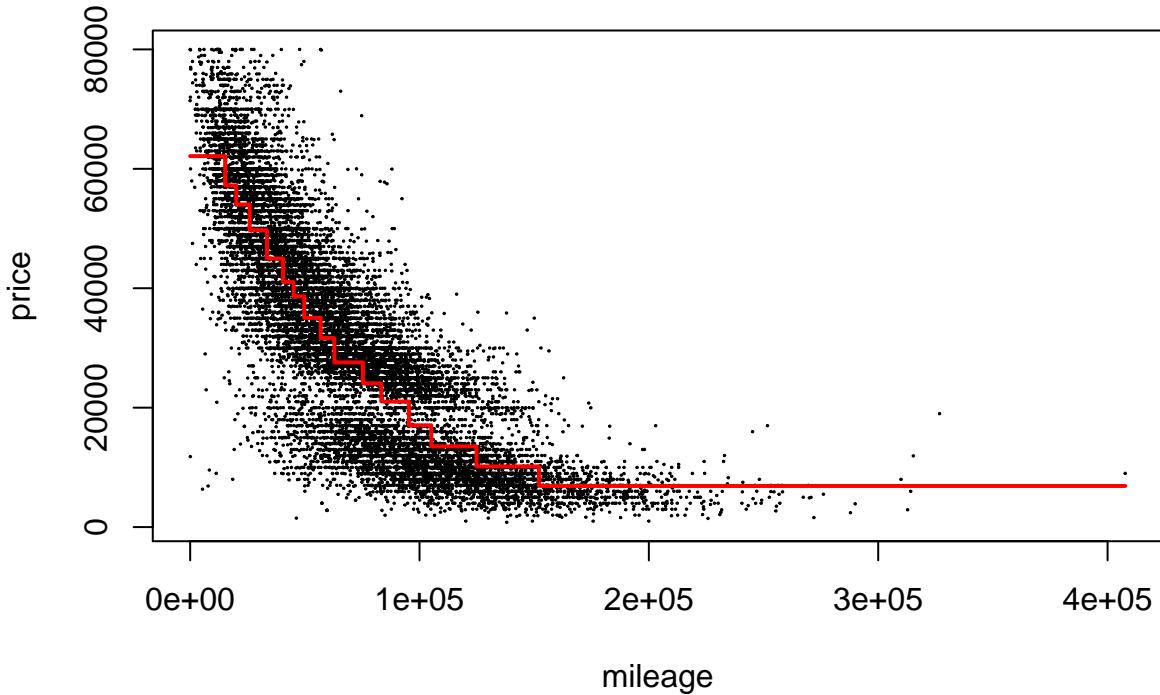
```

par(mfrow=c(1,1))
best.tree = prune(big.tree,cp=bestcp)
rpart.plot(best.tree)

```



```
plot(train$mileage, train$price, xlab = "mileage", ylab = "price", pch = 1, cex = 0.1)
lines(sort(train$mileage), predict(best.tree)[oo], col="red", lwd=2, cex.lab=2)
```



```
# error
sqrt(mean((test$price - predict(best.tree, test))^2))
```

```
## [1] 9545.132
```

Comparing with one variable case, now k of kNN and optimal tree size are smaller. Both method get lower RMSE with one more variable.

Part 7

```
big.tree = rpart(price~., data=train,
                  control=rpart.control(minsplit=5, cp=0.0001, xval=10))

nbig = length(unique(big.tree$where))
cat('size of big tree: ', nbig, '\n')

## size of big tree: 102
(cptable = printcp(big.tree))

##
## Regression tree:
## rpart(formula = price ~ ., data = train, control = rpart.control(minsplit = 5,
##           cp = 1e-04, xval = 10))
##
## Variables actually used in tree construction:
## [1] color          displacement mileage      region        soundSystem
## [6] trim          year
##
```

```

## Root node error: 5.008e+12/15047 = 332822195
##
## n= 15047
##
##          CP nsplit rel_error    xerror      xstd
## 1  0.61662322      0  1.000000 1.000109 0.0091413
## 2  0.17316315      1  0.383377 0.386580 0.0043738
## 3  0.05171357      2  0.210214 0.199123 0.0032176
## 4  0.01400228      3  0.158500 0.166033 0.0028922
## 5  0.01333573      4  0.144498 0.145898 0.0026398
## 6  0.00889372      5  0.131162 0.135061 0.0024662
## 7  0.00830282      6  0.122268 0.126808 0.0024102
## 8  0.00698175      7  0.113966 0.118509 0.0024009
## 9  0.00590622      8  0.106984 0.112548 0.0022896
## 10 0.00392076      9  0.101078 0.104663 0.0021261
## 11 0.00379582     10  0.097157 0.100542 0.0020598
## 12 0.00334702     11  0.093361 0.098052 0.0020281
## 13 0.00238000     12  0.090014 0.091071 0.0019290
## 14 0.00165064     13  0.087634 0.088127 0.0018983
## 15 0.00165031     14  0.085983 0.086531 0.0019040
## 16 0.00157980     15  0.084333 0.085890 0.0018872
## 17 0.00156529     16  0.082753 0.084914 0.0018728
## 18 0.00156279     17  0.081188 0.084615 0.0018703
## 19 0.00121171     18  0.079625 0.082570 0.0018387
## 20 0.00120888     19  0.078413 0.080298 0.0017906
## 21 0.00108720     21  0.075996 0.079212 0.0017739
## 22 0.00101711     22  0.074908 0.078106 0.0017524
## 23 0.00082202     23  0.073891 0.076294 0.0017026
## 24 0.00077895     24  0.073069 0.075108 0.0016752
## 25 0.00076646     25  0.072290 0.074394 0.0016570
## 26 0.00064861     26  0.071524 0.073449 0.0016505
## 27 0.00064715     27  0.070875 0.072196 0.0016269
## 28 0.00063162     28  0.070228 0.071891 0.0016226
## 29 0.00063107     29  0.068965 0.071707 0.0016221
## 30 0.00061136     30  0.068334 0.071707 0.0016221
## 31 0.00051406     31  0.067722 0.070558 0.0016065
## 32 0.00050215     32  0.067208 0.070050 0.0016059
## 33 0.00050123     33  0.066706 0.069678 0.0016374
## 34 0.00049495     34  0.066205 0.069600 0.0016367
## 35 0.00043115     35  0.065710 0.068836 0.0016305
## 36 0.00042389     36  0.065279 0.068174 0.0016229
## 37 0.00042116     37  0.064855 0.068043 0.0016223
## 38 0.00041973     38  0.064434 0.067894 0.0016219
## 39 0.00041122     39  0.064014 0.067699 0.0016196
## 40 0.00040802     40  0.063603 0.067316 0.0016169
## 41 0.00037489     41  0.063195 0.066976 0.0016143
## 42 0.00036827     42  0.062820 0.065966 0.0016074
## 43 0.00036290     43  0.062083 0.065665 0.0016019
## 44 0.00034924     44  0.061721 0.065305 0.0015930
## 45 0.00033263     45  0.061371 0.064702 0.0015889
## 46 0.00031679     46  0.060706 0.064416 0.0015811
## 47 0.00026252     47  0.060389 0.064176 0.0016018
## 48 0.00024995     48  0.060127 0.063890 0.0016113
## 49 0.00024629     49  0.059877 0.063896 0.0016114

```

```

## 50 0.00024597      53 0.059630 0.063932 0.0016114
## 51 0.00023915      54 0.059385 0.063800 0.0016108
## 52 0.00023886      55 0.059145 0.063777 0.0016101
## 53 0.00023806      56 0.058907 0.063735 0.0016098
## 54 0.00023748      57 0.058668 0.063735 0.0016098
## 55 0.00021534      58 0.058431 0.063533 0.0016100
## 56 0.00020969      59 0.058216 0.063050 0.0016063
## 57 0.00020357      60 0.058006 0.062801 0.0016054
## 58 0.00020243      61 0.057802 0.062733 0.0016051
## 59 0.00020165      62 0.057600 0.062677 0.0016049
## 60 0.00020060      63 0.057398 0.062651 0.0016044
## 61 0.00019943      64 0.057198 0.062599 0.0016042
## 62 0.00019856      65 0.056998 0.062562 0.0016043
## 63 0.00019494      66 0.056800 0.062444 0.0016040
## 64 0.00018167      67 0.056605 0.062094 0.0016028
## 65 0.00017622      68 0.056423 0.061543 0.0015973
## 66 0.00017249      69 0.056247 0.061328 0.0015964
## 67 0.00017024      70 0.056074 0.061285 0.0015980
## 68 0.00015539      71 0.055904 0.061182 0.0015872
## 69 0.00014705      72 0.055749 0.060708 0.0015814
## 70 0.00014651      73 0.055602 0.060516 0.0015789
## 71 0.00014595      74 0.055455 0.060501 0.0015789
## 72 0.00014485      75 0.055309 0.060477 0.0015788
## 73 0.00013859      76 0.055164 0.060588 0.0015806
## 74 0.00013746      77 0.055026 0.060573 0.0015854
## 75 0.00013527      78 0.054888 0.060506 0.0015842
## 76 0.00013153      79 0.054753 0.060511 0.0015850
## 77 0.00012782      80 0.054622 0.060493 0.0015864
## 78 0.00012754      81 0.054494 0.060450 0.0015862
## 79 0.00012517      82 0.054366 0.060419 0.0015868
## 80 0.00012393      83 0.054241 0.060378 0.0015868
## 81 0.00012384      84 0.054117 0.060315 0.0015860
## 82 0.00011745      85 0.053993 0.060149 0.0015848
## 83 0.00011678      86 0.053876 0.060138 0.0015863
## 84 0.00011466      87 0.053759 0.060123 0.0015869
## 85 0.00011195      88 0.053644 0.060066 0.0015878
## 86 0.00011096      89 0.053420 0.060192 0.0015992
## 87 0.00011011      90 0.053088 0.060208 0.0015995
## 88 0.00010970      91 0.052977 0.060261 0.0016010
## 89 0.00010620      92 0.052868 0.060275 0.0016021
## 90 0.00010612      93 0.052762 0.060308 0.0016021
## 91 0.00010499      94 0.052655 0.060299 0.0016020
## 92 0.00010375      95 0.052550 0.060318 0.0016021
## 93 0.00010284      96 0.052447 0.060302 0.0016020
## 94 0.00010257      97 0.052344 0.060337 0.0016027
## 95 0.00010000      98 0.052241 0.060204 0.0015955

##          CP nsplit rel.error      xerror      xstd
## 1 0.6166232237    0 1.00000000 1.00010894 0.009141261
## 2 0.1731631458    1 0.38337678 0.38657986 0.004373801
## 3 0.0517135744    2 0.21021363 0.19912337 0.003217621
## 4 0.0140022783    3 0.15850006 0.16603285 0.002892206
## 5 0.0133357270    4 0.14449778 0.14589817 0.002639773
## 6 0.0088937244    5 0.13116205 0.13506067 0.002466198

```

```

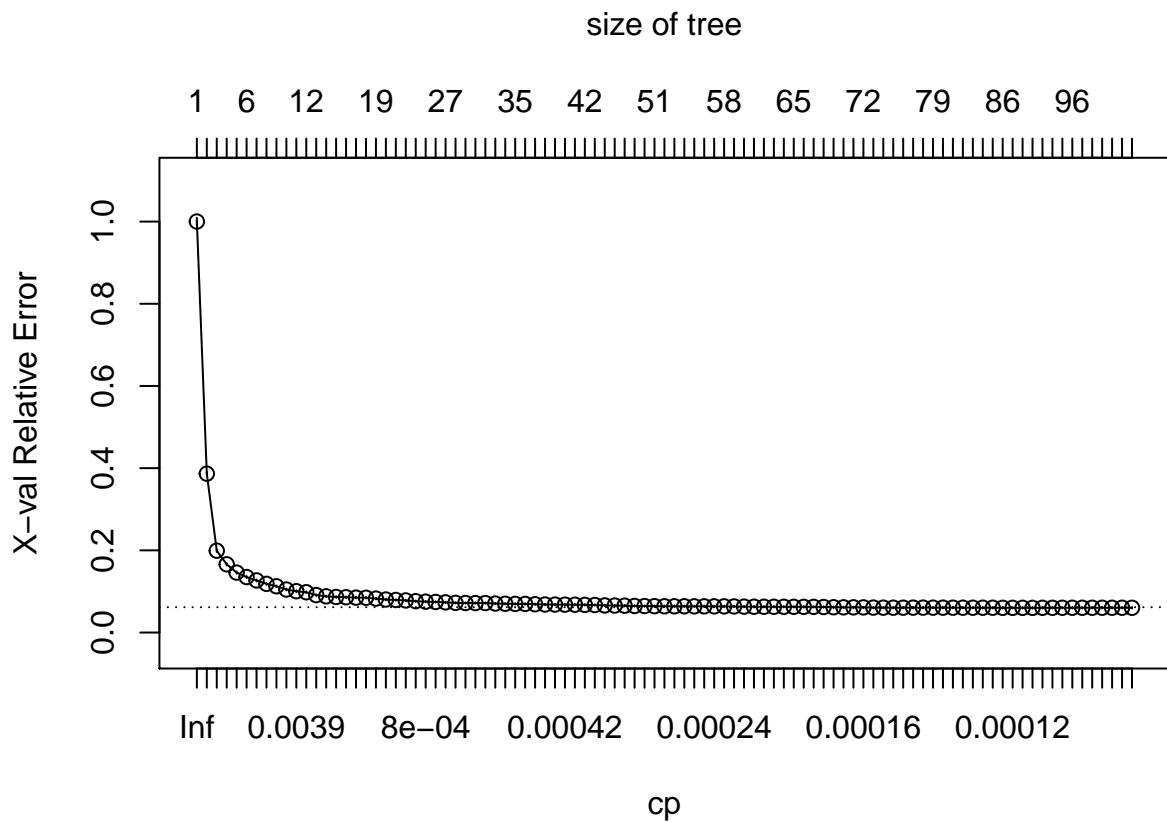
## 7  0.0083028208      6  0.12226833  0.12680846  0.002410218
## 8  0.0069817538      7  0.11396551  0.11850926  0.002400890
## 9  0.0059062247      8  0.10698375  0.11254840  0.002289574
## 10 0.0039207634     9  0.10107753  0.10466334  0.002126093
## 11 0.0037958238    10 0.09715676  0.10054233  0.002059784
## 12 0.0033470204    11 0.09336094  0.09805230  0.002028058
## 13 0.0023800029    12 0.09001392  0.09107132  0.001928964
## 14 0.0016506425    13 0.08763392  0.08812712  0.001898294
## 15 0.0016503086    14 0.08598327  0.08653149  0.001903969
## 16 0.0015798041    15 0.08433297  0.08588952  0.001887163
## 17 0.0015652863    16 0.08275316  0.08491373  0.001872834
## 18 0.0015627898    17 0.08118787  0.08461469  0.001870336
## 19 0.0012117063    18 0.07962509  0.08257024  0.001838674
## 20 0.0012088800    19 0.07841338  0.08029771  0.001790634
## 21 0.0010872031    21 0.07599562  0.07921189  0.001773908
## 22 0.0010171066    22 0.07490842  0.07810645  0.001752361
## 23 0.0008220185    23 0.07389131  0.07629371  0.001702584
## 24 0.0007789480    24 0.07306929  0.07510752  0.001675188
## 25 0.0007664638    25 0.07229034  0.07439446  0.001656980
## 26 0.0006486098    26 0.07152388  0.07344922  0.001650486
## 27 0.0006471527    27 0.07087527  0.07219649  0.001626895
## 28 0.0006316216    28 0.07022812  0.07189148  0.001622597
## 29 0.0006310691    29 0.06896487  0.07170724  0.001622119
## 30 0.0006113592    30 0.06833380  0.07170724  0.001622119
## 31 0.0005140579    31 0.06772244  0.07055842  0.001606542
## 32 0.0005021461    32 0.06720839  0.07005013  0.001605866
## 33 0.0005012348    33 0.06670624  0.06967840  0.001637387
## 34 0.0004949475    34 0.06620501  0.06960037  0.001636684
## 35 0.0004311512    35 0.06571006  0.06883566  0.001630478
## 36 0.0004238851    36 0.06527891  0.06817435  0.001622862
## 37 0.0004211600    37 0.06485502  0.06804313  0.001622317
## 38 0.0004197348    38 0.06443386  0.06789383  0.001621858
## 39 0.0004112215    39 0.06401413  0.06769929  0.001619620
## 40 0.0004080176    40 0.06360291  0.06731579  0.001616906
## 41 0.0003748905    41 0.06319489  0.06697575  0.001614312
## 42 0.0003682713    42 0.06282000  0.06596610  0.001607411
## 43 0.0003628976    43 0.06208346  0.06566510  0.001601903
## 44 0.0003492431    44 0.06172056  0.06530490  0.001593040
## 45 0.0003326344    45 0.06137131  0.06470169  0.001588893
## 46 0.0003167887    46 0.06070605  0.06441628  0.001581116
## 47 0.0002625241    47 0.06038926  0.06417602  0.001601761
## 48 0.0002499452    48 0.06012673  0.06388982  0.001611282
## 49 0.0002462906    49 0.05987679  0.06389612  0.001611383
## 50 0.0002459669    50 0.05963050  0.06393228  0.001611444
## 51 0.0002391502    51 0.05938453  0.06380040  0.001610768
## 52 0.0002388580    52 0.05914538  0.06377670  0.001610066
## 53 0.0002380568    53 0.05890652  0.06373504  0.001609767
## 54 0.0002374792    54 0.05866847  0.06373504  0.001609767
## 55 0.0002153422    55 0.05843099  0.06353306  0.001609964
## 56 0.0002096887    56 0.05821564  0.06305050  0.001606263
## 57 0.0002035727    57 0.05800595  0.06280053  0.001605406
## 58 0.0002024271    58 0.05780238  0.06273279  0.001605062
## 59 0.0002016532    59 0.05759996  0.06267657  0.001604932
## 60 0.0002006046    60 0.05739830  0.06265083  0.001604384

```

```

## 61 0.0001994274      64 0.05719770 0.06259863 0.001604172
## 62 0.0001985642      65 0.05699827 0.06256205 0.001604308
## 63 0.0001949440      66 0.05679971 0.06244393 0.001603995
## 64 0.0001816657      67 0.05660476 0.06209374 0.001602798
## 65 0.0001762215      68 0.05642310 0.06154335 0.001597323
## 66 0.0001724918      69 0.05624687 0.06132847 0.001596424
## 67 0.0001702403      70 0.05607438 0.06128473 0.001598030
## 68 0.0001553936      71 0.05590414 0.06118182 0.001587189
## 69 0.0001470496      72 0.05574875 0.06070774 0.001581411
## 70 0.0001465080      73 0.05560170 0.06051559 0.001578906
## 71 0.0001459511      74 0.05545519 0.06050059 0.001578901
## 72 0.0001448544      75 0.05530924 0.06047706 0.001578792
## 73 0.0001385946      76 0.05516439 0.06058809 0.001580629
## 74 0.0001374574      77 0.05502579 0.06057298 0.001585438
## 75 0.0001352742      78 0.05488833 0.06050558 0.001584172
## 76 0.0001315331      79 0.05475306 0.06051140 0.001585038
## 77 0.0001278190      80 0.05462153 0.06049264 0.001586433
## 78 0.0001275400      81 0.05449371 0.06044975 0.001586166
## 79 0.0001251691      82 0.05436617 0.06041898 0.001586791
## 80 0.0001239268      83 0.05424100 0.06037812 0.001586771
## 81 0.0001238351      84 0.05411707 0.06031548 0.001586045
## 82 0.0001174523      85 0.05399324 0.06014890 0.001584807
## 83 0.0001167788      86 0.05387578 0.06013846 0.001586323
## 84 0.0001146643      87 0.05375901 0.06012339 0.001586890
## 85 0.0001119541      88 0.05364434 0.06006562 0.001587803
## 86 0.0001109636      89 0.05342043 0.06019209 0.001599240
## 87 0.0001101065      90 0.05308754 0.06020829 0.001599458
## 88 0.0001097049      91 0.05297744 0.06026117 0.001601028
## 89 0.0001061964      92 0.05286773 0.06027470 0.001602093
## 90 0.0001061154      93 0.05276153 0.06030830 0.001602083
## 91 0.0001049873      94 0.05265542 0.06029884 0.001601989
## 92 0.0001037524      95 0.05255043 0.06031780 0.001602069
## 93 0.0001028405      96 0.05244668 0.06030226 0.001601987
## 94 0.0001025679      97 0.05234384 0.06033734 0.001602654
## 95 0.0001000000      98 0.05224127 0.06020438 0.001595532
(bestcp = cptable[ which.min(cptable[, "xerror"]), "CP" ]) # this is the optimal cp parameter
## [1] 0.0001119541
plotcp(big.tree) # plot results

```



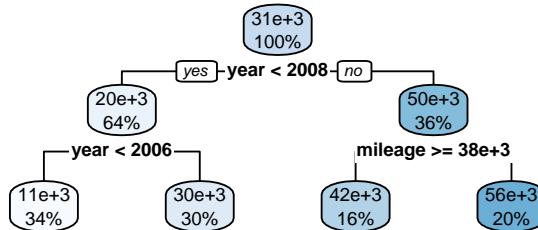
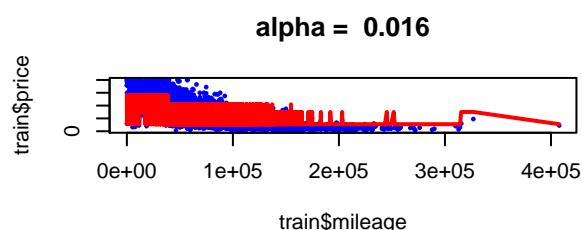
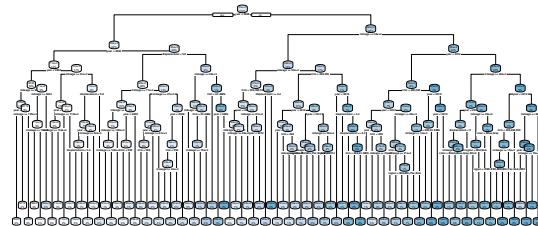
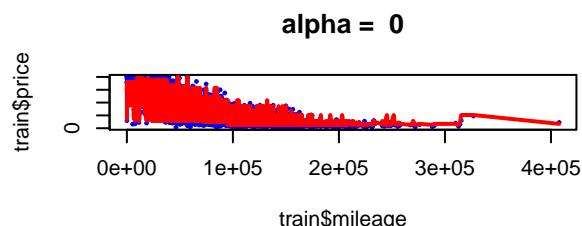
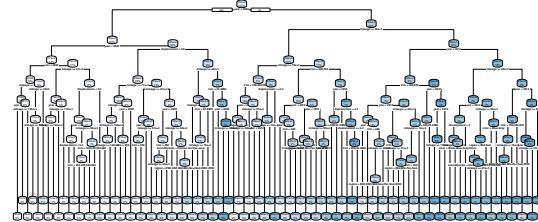
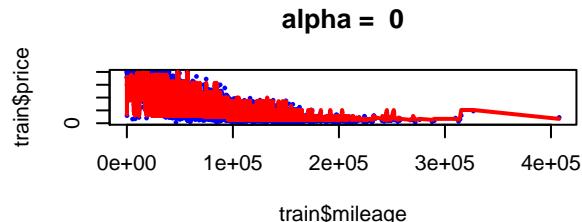
```

# show fit from some trees
oo = order(train$mileage)
cpvec = c(bestcp / 2, bestcp,.0157)

par(mfrow=c(3,2))
for(i in 1:3) {
  plot(train$mileage,train$price,pch=16,col='blue',cex=.5)
  ptree = prune(big.tree,cp=cpvec[i])
  pfit = predict(ptree)
  lines(train$mileage[oo],pfit[oo],col='red',lwd=2)
  title(paste('alpha = ',round(cpvec[i],3)))
  rpart.plot(ptree)
}

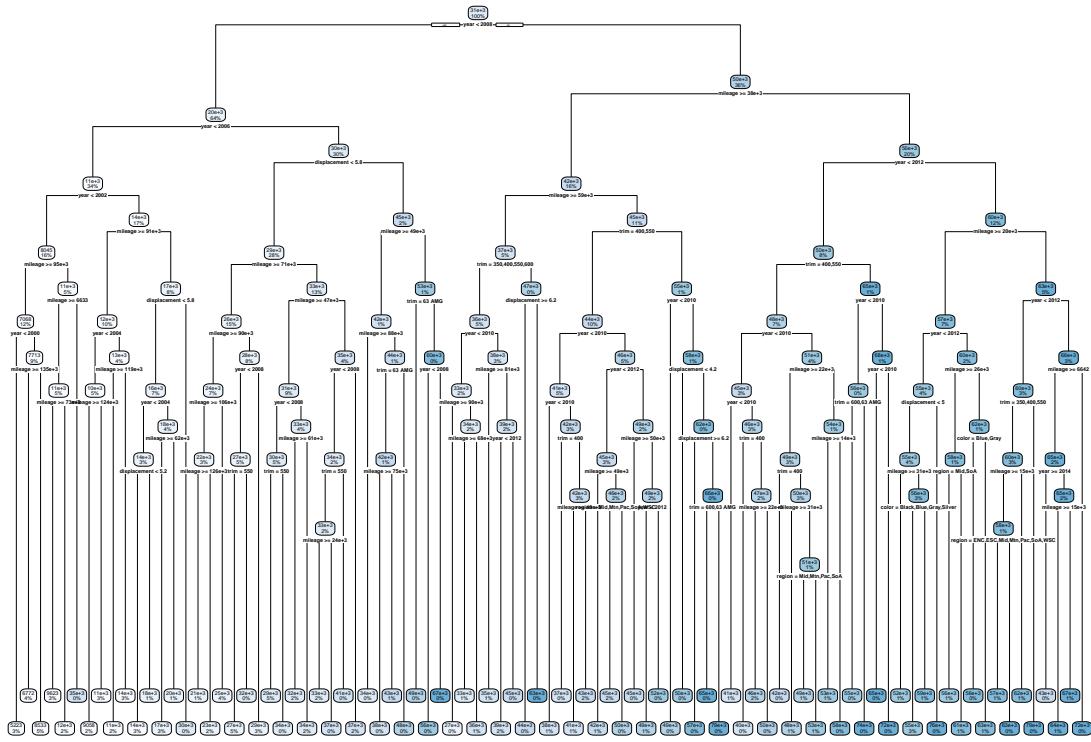
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
## Warning: labs do not fit even at cex 0.15, there may be some overplotting

```

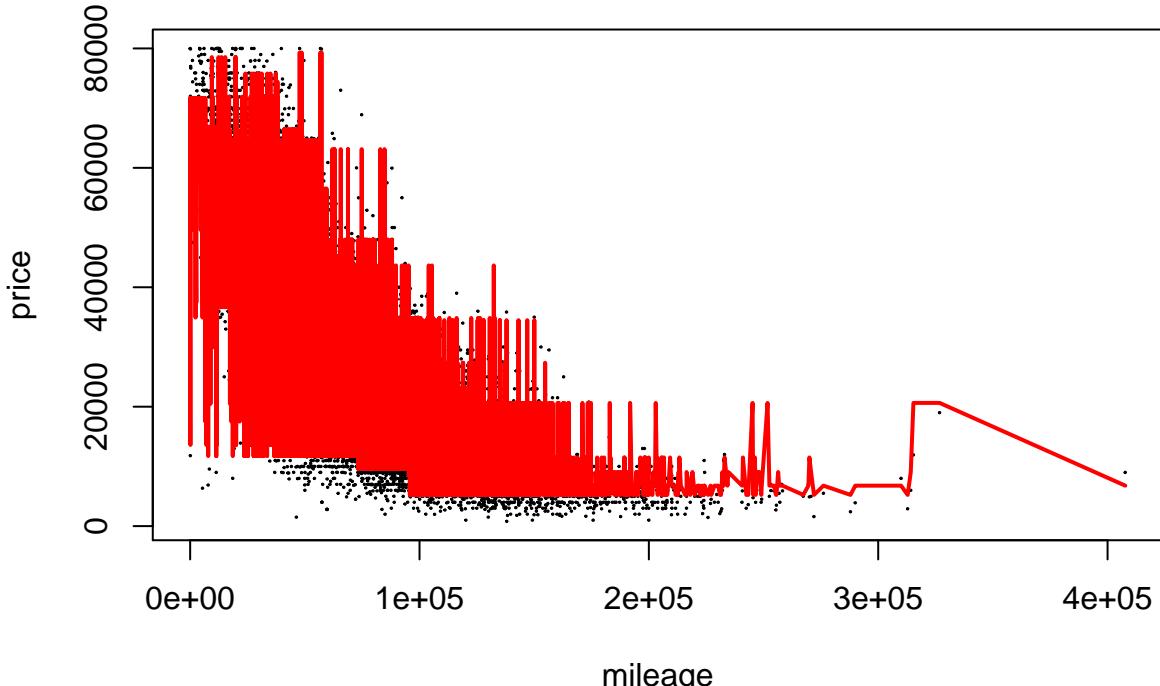


```
par(mfrow=c(1,1))
best.tree = prune(big.tree, cp=bestcp)
rpart.plot(best.tree)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



```
plot(train$mileage, train$price, xlab = "mileage", ylab = "price", pch = 1, cex = 0.1)
lines(sort(train$mileage), predict(best.tree)[oo], col="red", lwd=2, cex.lab=2)
```



```
# error  
sqrt(mean((test$price - predict(best.tree, test))^2))
```

```
## [1] 4490.279
```

Bonus

One possible procedure

- For each fixed k , there are $\binom{p}{k}$ possible k -variable models. Run cross validation and compute RMSE for all possible models. The model with smallest RMSE is the optimal k -variable model.
- For all $k = 1, \dots, p$. Compare RMSE of all optimal k -variable models and find the one with smallest RMSE.