Homework 1 Solution

Question 1

1.1

As one increases k, the number of nearest neighbor, in a kNN classifier:

(a) **TRUE**. With more neighbors, the prediction is less localized. (This concept is intended to mean over all conceptual datasets.)

(b) **FALSE**. With more neighbors, your predictions will become more stable. (Again, over all conceptual datasets.)

(c) **FALSE**. In general there is a upward trend, but only stochastically. It might sometimes drop a bit. (Due to the randomness of the data and the non-smooth property of kNN)

(d) **FALSE**. There might be some optimal k that reaches a minimum test error, but we have no prior information if we are below or above it.

1.2

Consider the three line regression fits to the gray points plotted below.

(a) **TRUE**. More parameters mean more fluctuation in predictions.

(b) **TRUE**. More parameters lead to less bias, although model (3) looks potentially overfitting.

(c) **TRUE**. Model (3) has the most degrees of freedom fot thr best training fit. (Think this way: model (2) can't beat model (3) for training data, since model (3) can at the very least get an easy draw by just matching $\beta_0, \beta_1, \beta_2$ to model (2) and setting β_j for $j \geq 3$ to zero.)

(d) **FALSE**. Without test data, we are nowhere near a conclusion which model has the smallest test error. (We CANNOT even say (2) has the smallest test error, in two ways: first, if the sample size is too small, the randomness in the data might dominate, and the training set itself might not represent the underlying relation; second, even with a sufficient sample size, the underlying data-generation scheme might truly be a 20-dimensional polynomial, who knows.)

1.3

FALSE. The misclassification rate calculated based on a dataset is always subject to the effect of the randomness of that dataset. Despite being unlikely, the error on the validation set could still be lower than that on the training set. (e.g. in the unlikely event where we built the classifier with the toughest cases but the validation data contained just some regular data)

1.4

FALSE. It is slightly biased upwards. Training with a slightly smaller set than the full dataset, the fit isn't quite as good as if you'd used the full set for each fold. Thus *on average* the error calculated on the heldout fraction is slightly higher.

Question 2

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=0.1)
  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)
}
```

2.1 - 2.5. Linear Model

Define function

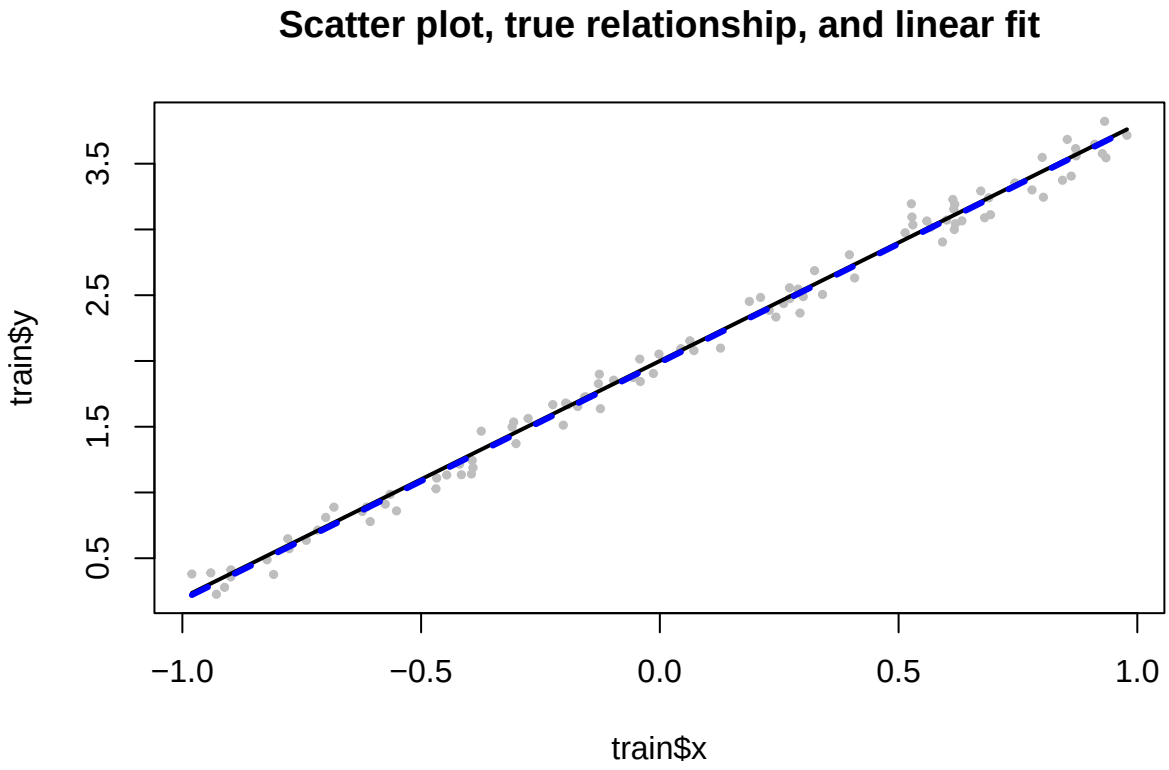
```
func_linear <- function(x){1.8*x+2}
```

Simulate data from the linear function

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

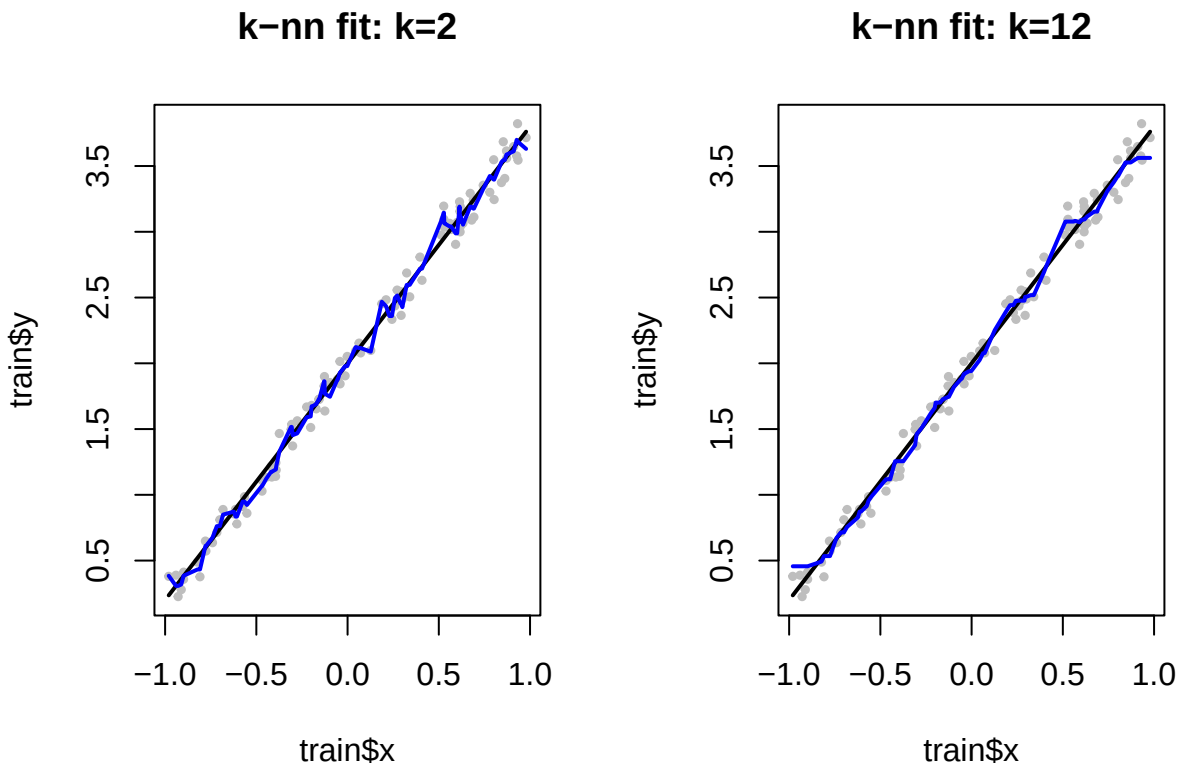
Plot the training dataset, true relationship and best linear fit

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



Plot kNN fit with k=2 and k=12 respectively

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



The prediction under $k = 2$ is apparantly more fluctuated than the one onder $k = 12$. For a very small k value, the model has a high variance and is prone to overfitting. A larger k value reduces variance at the cost

of a larger bias (e.g. around $x = \pm 1.0$).

Plot test set MSE

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

- Observation

Linear regression model outperforms kNN 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, kNN performs only a little worse than least square regression. It does far worse when k is small.

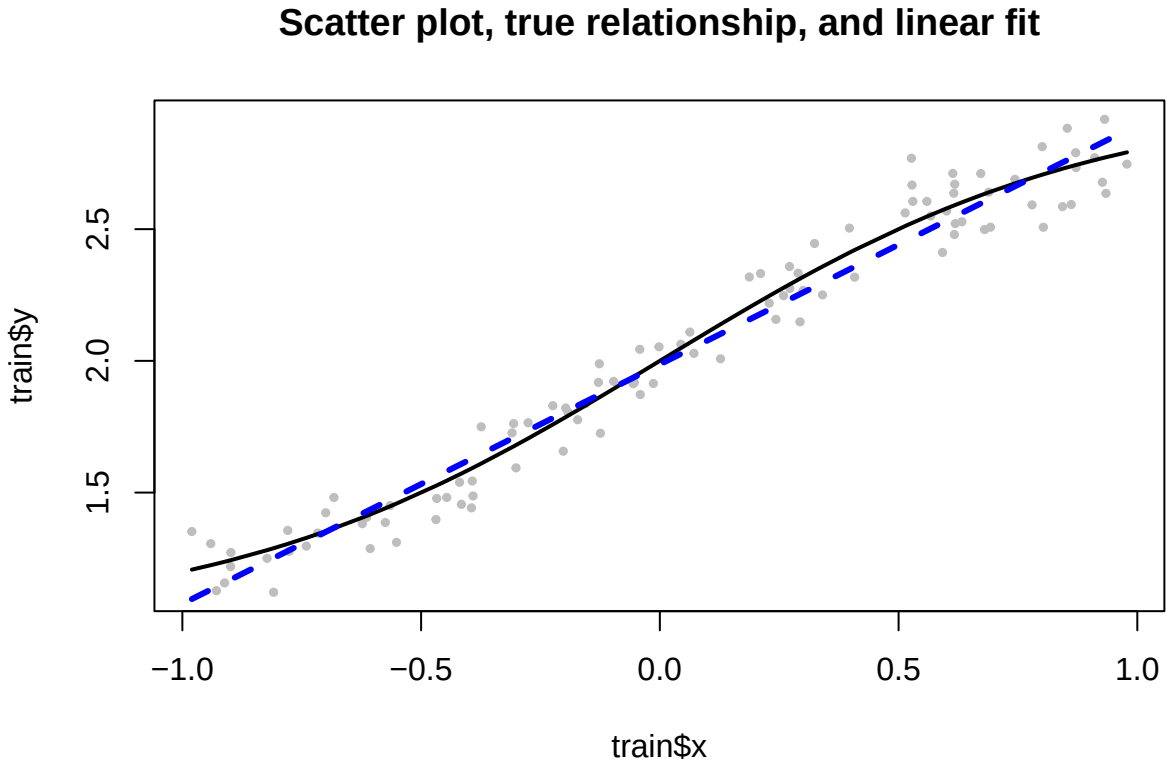
2.6. Almost Linear Model

Define function

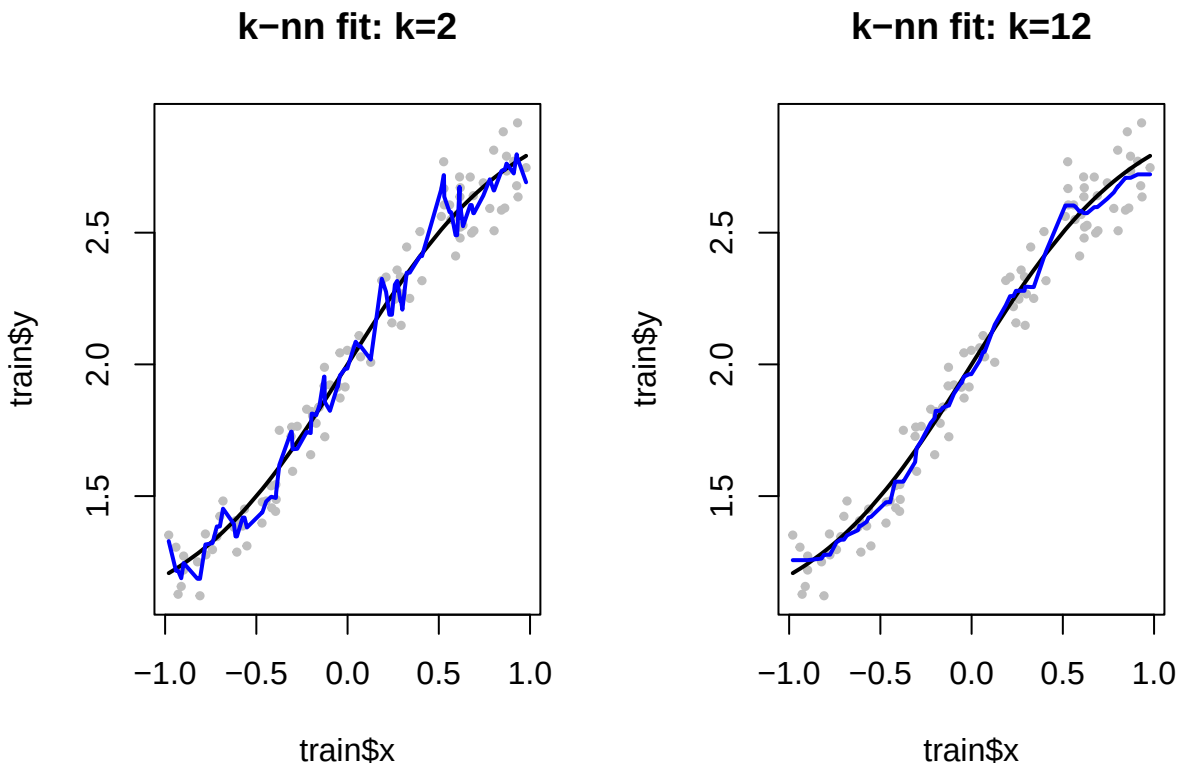
```
func_almost_linear <- function(x){tanh(1.1*x)+2}
```

Simulate data and plot

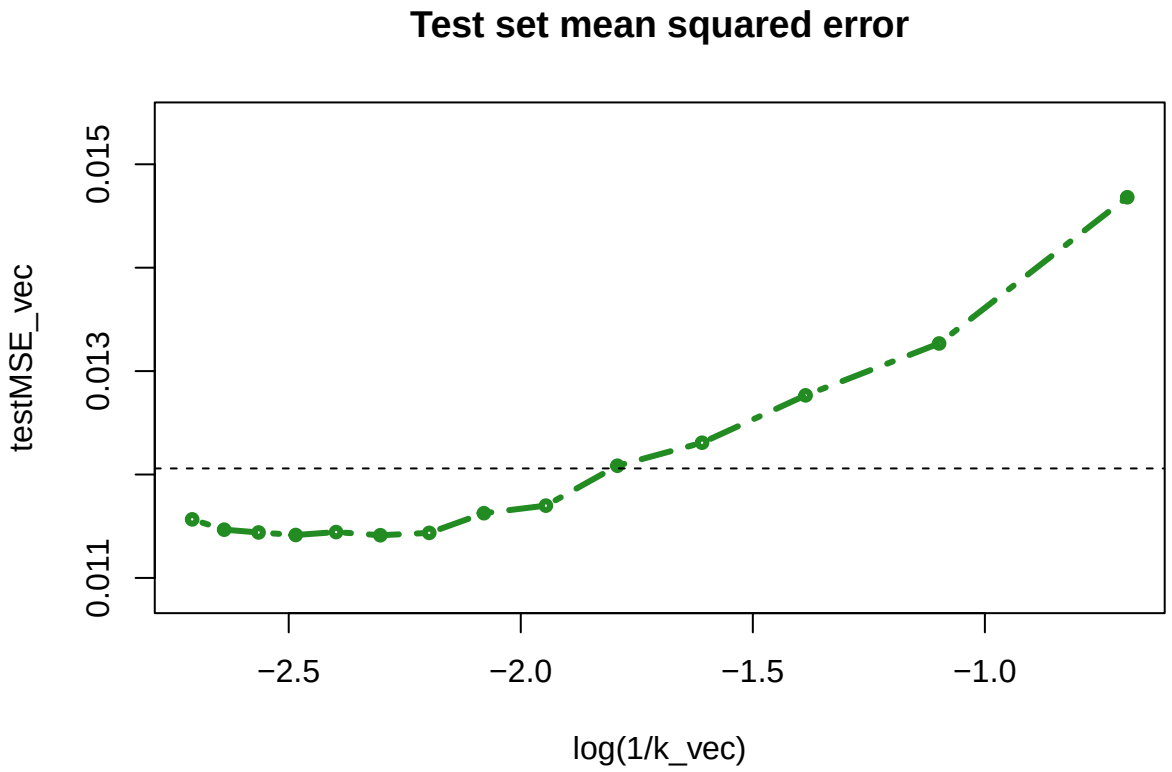
```
sim_aslinear <- Simulate(func_almost_linear, N_train, N_test)
plot_ls_aslinear <- Plot(sim_aslinear, method='linear regression')
```




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




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



- Observation

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

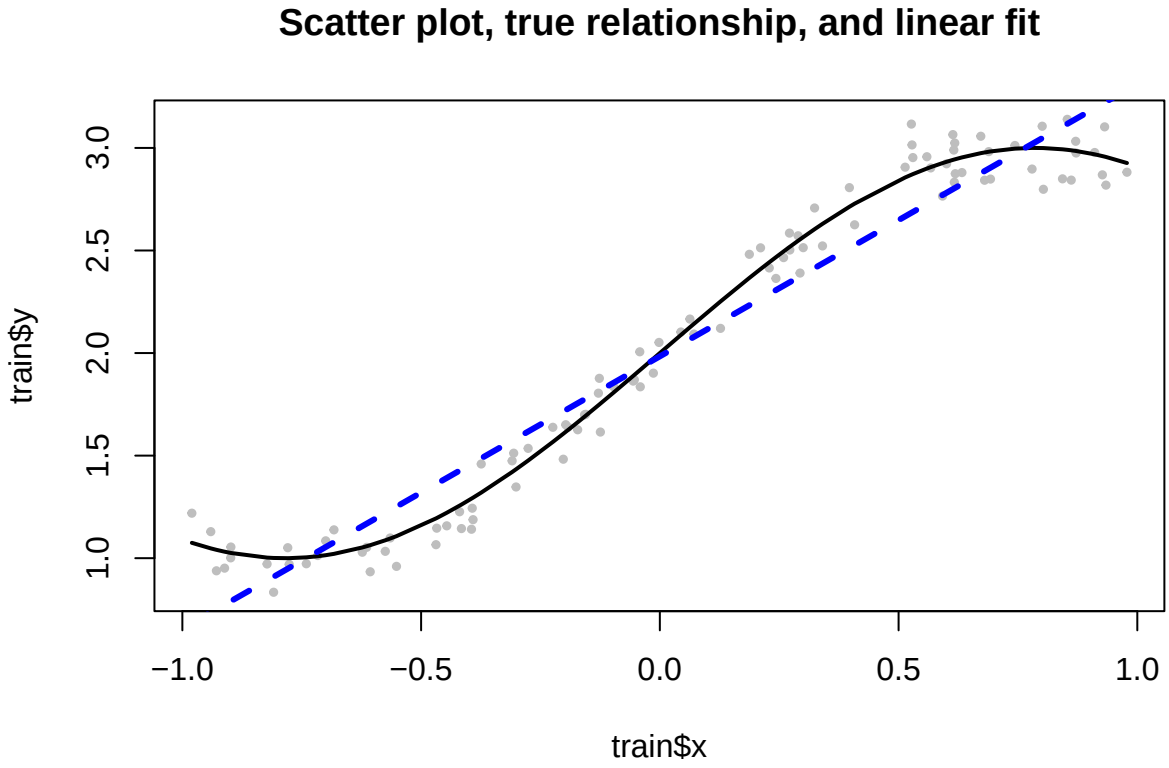
2.7. Strongly Nonlinear Model

Define function

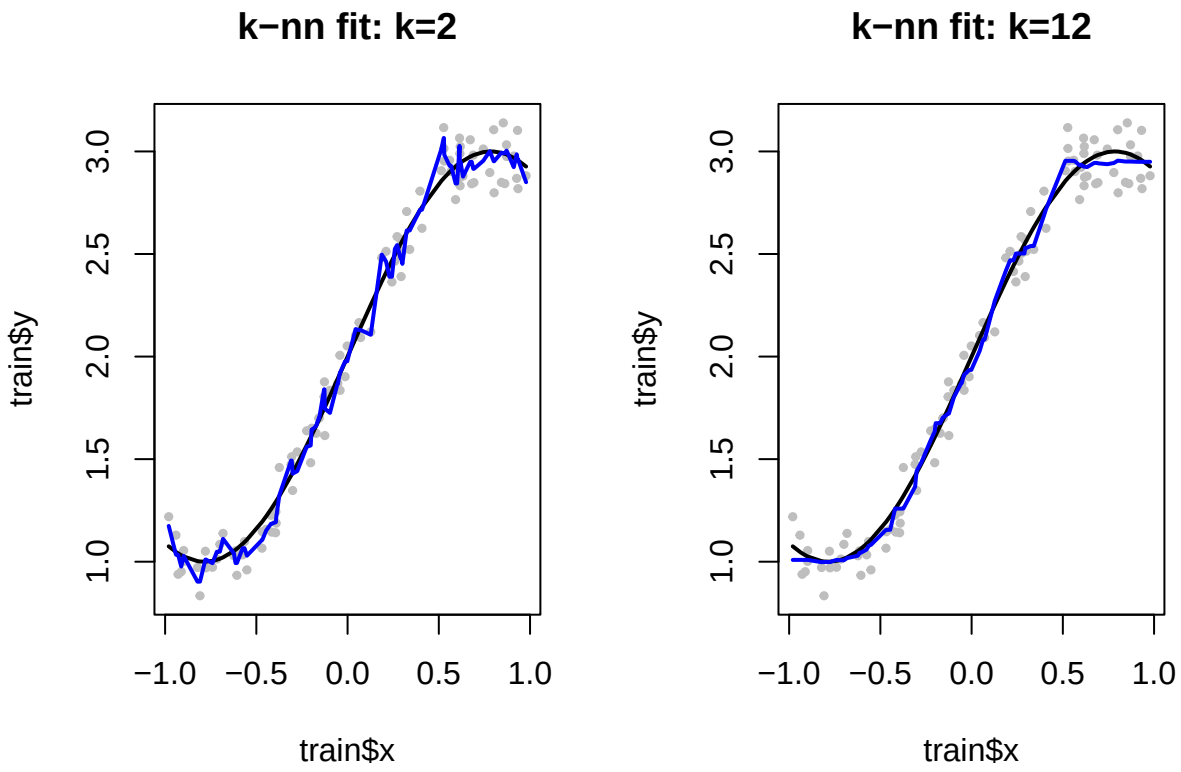
```
func_non_linear <- function(x){sin(2*x)+2}
```

Simulate data and plot

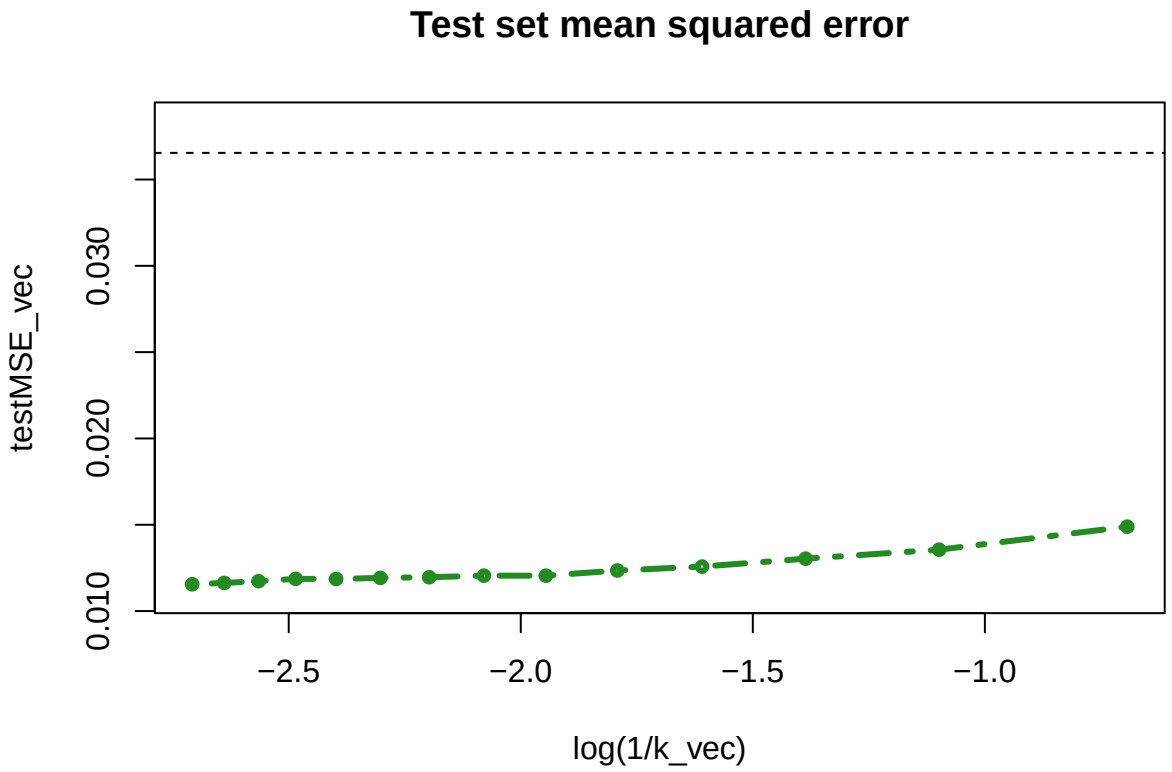
```
sim_nlinear <- Simulate(func_non_linear, N_train, N_test)
plot_ls_nlinear <- Plot(sim_nlinear, method='linear regression')
```




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




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



- Observation

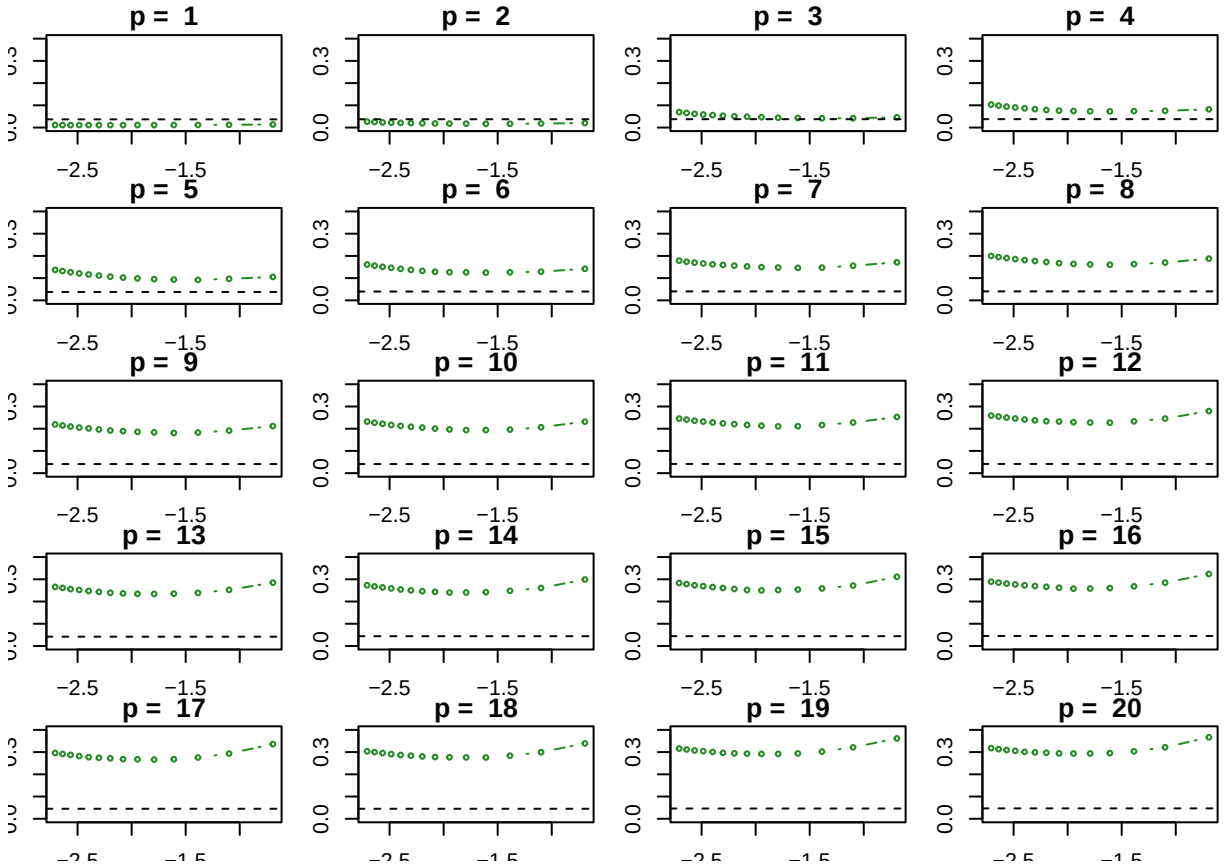
When the true relationship is highly non-linear, kNN gives much better result than linear regression, for all values of k. The test MSE of linear regression increases dramatically with the extent of non-linearity while that of kNN changes little.

2.8. Multiple Predictors

```
set.seed(233)
Xtrain=matrix(c(runif(N_train,-1,1),runif(N_train*19)), nrow=N_train)
Xtest=matrix(c(runif(N_test,-1,1),runif(N_test*19)), nrow=N_test)
Ytrain=sin(2*Xtrain[,1])+2+rnorm(N_train, sd=0.1)
Ytest=sin(2*Xtest[,1])+2+rnorm(N_test, sd=0.1)
```

```
par(mfrow = c(5, 4), mai=c(0.2,0.2,0.2,0.2))
for (p in 1:20) {
  train=data.frame(Ytrain, Xtrain[,1:p])
  test=data.frame(Ytest, Xtest[,1:p])
  colnames(test) <- colnames(train) <- c("y", paste0("x", 1:p))
  linear=lm(y~., data=train)
  lsMSE=mean((predict(linear, test)-test$y)^2)
  k_vec=2:15
  testMSE_vec=numeric(length=length(k_vec))
  for (i in 1:length(k_vec)){
    near=kknn(y~., train=train, test=test, k=k_vec[i], kernel='rectangular')
    testMSE_vec[i]=mean((near$fitted-test$y)^2)
  }
  plot(log(1/k_vec), testMSE_vec,
       lty=6, lwd=1, col="forestgreen", type="b", cex=0.5,
       ylim = c(0,0.4),main=paste("p = ",p))
```

```
  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 kNN. 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 kNN method when there is a small number of observations per predictor. This trend basically holds in our simulation, though not monotonically.

2.9. Increasing the sample size

- Our conclusion concerning the relative performance of linear regression and kNN as the extent of model non-linearity varies will remain unchanged.
- We will have a wider range of values of k for which kNN 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 kNN. In other words, kNN 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 3

3.1

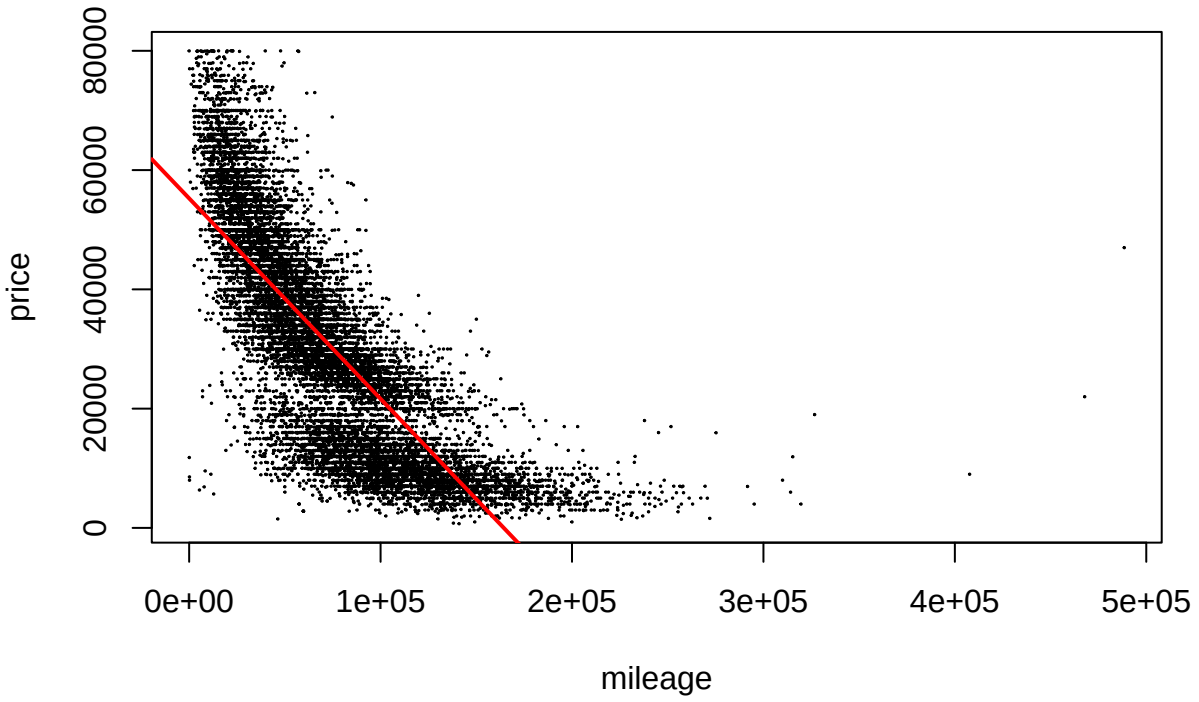
```
# set seed
set.seed(100)

UsedCars <- read.csv(file="UsedCars.csv",head=TRUE,sep=",")
```

This is a dataset that shows the price of 20,063 used cars and their information. The data is collected to evaluate the price of a used car based on its mileage, year, and displacement, etc.

3.2

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

```
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.

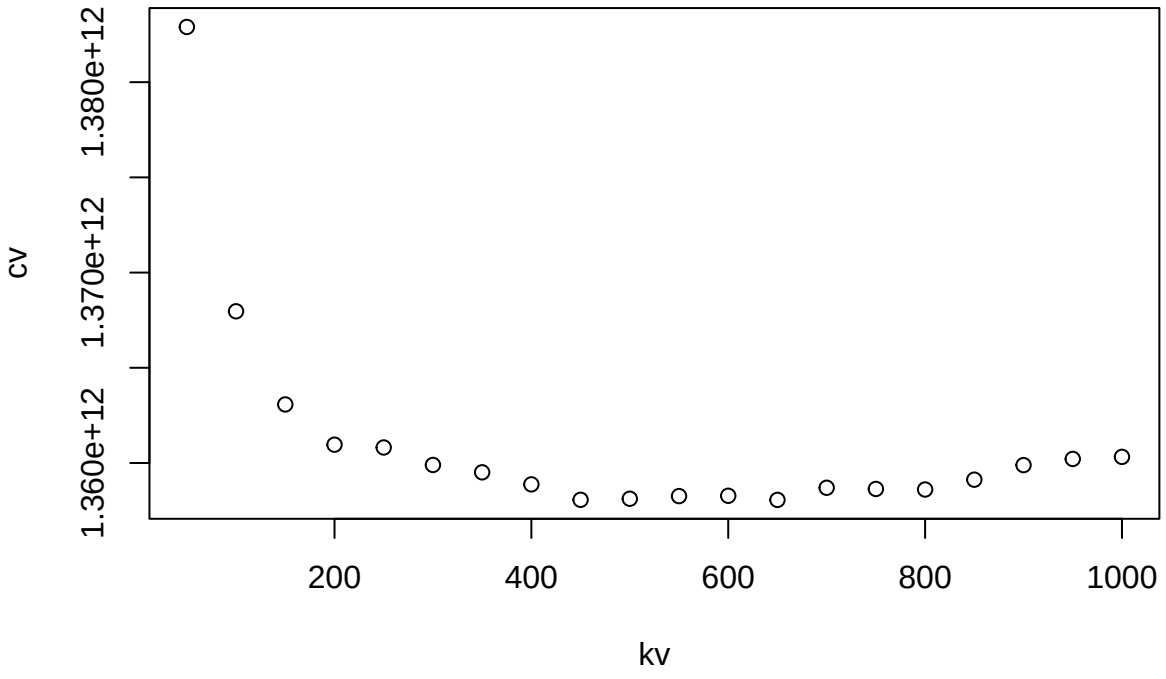
3.3

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

```
##########

library(MASS)
library(kknn)
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)
```

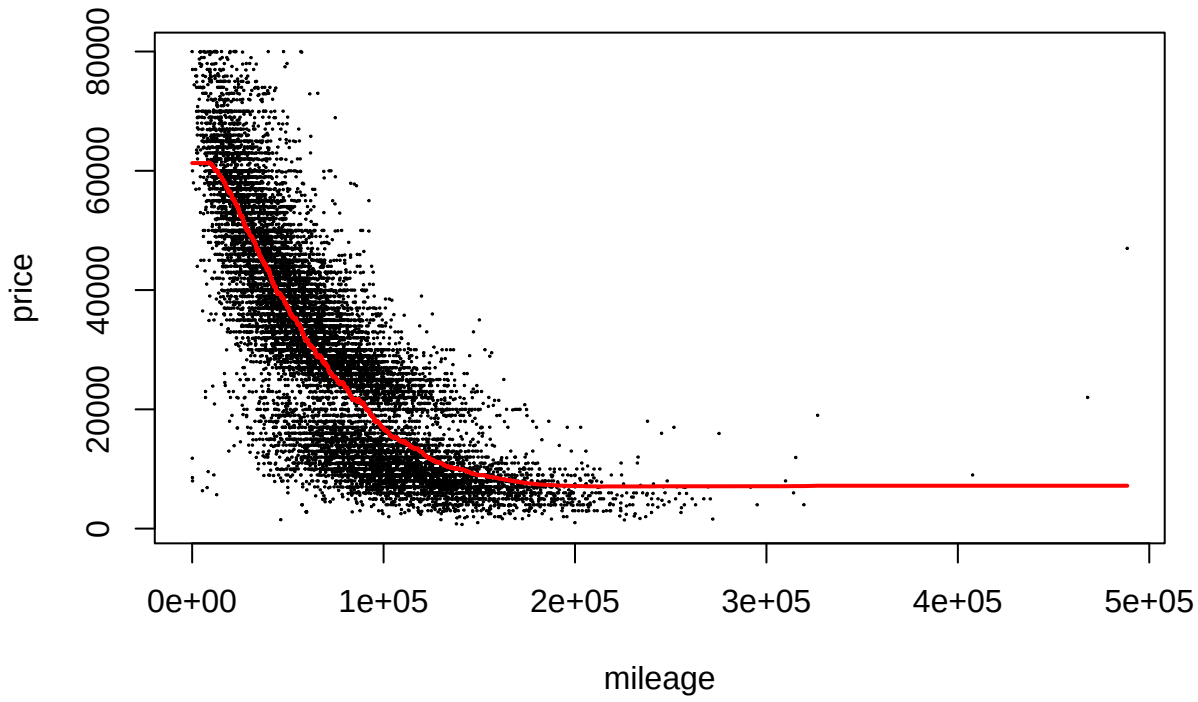
```
## in docv: nset,n,nfold:  20 15047 10
## on fold:  1 , range:  1 : 1505
## on fold:  2 , range:  1506 : 3010
## on fold:  3 , range:  3011 : 4515
## on fold:  4 , range:  4516 : 6020
## on fold:  5 , range:  6021 : 7525
## on fold:  6 , range:  7526 : 9030
## on fold:  7 , range:  9031 : 10535
##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

```
plot(kv, cv)
```




```
kbest = 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)
```



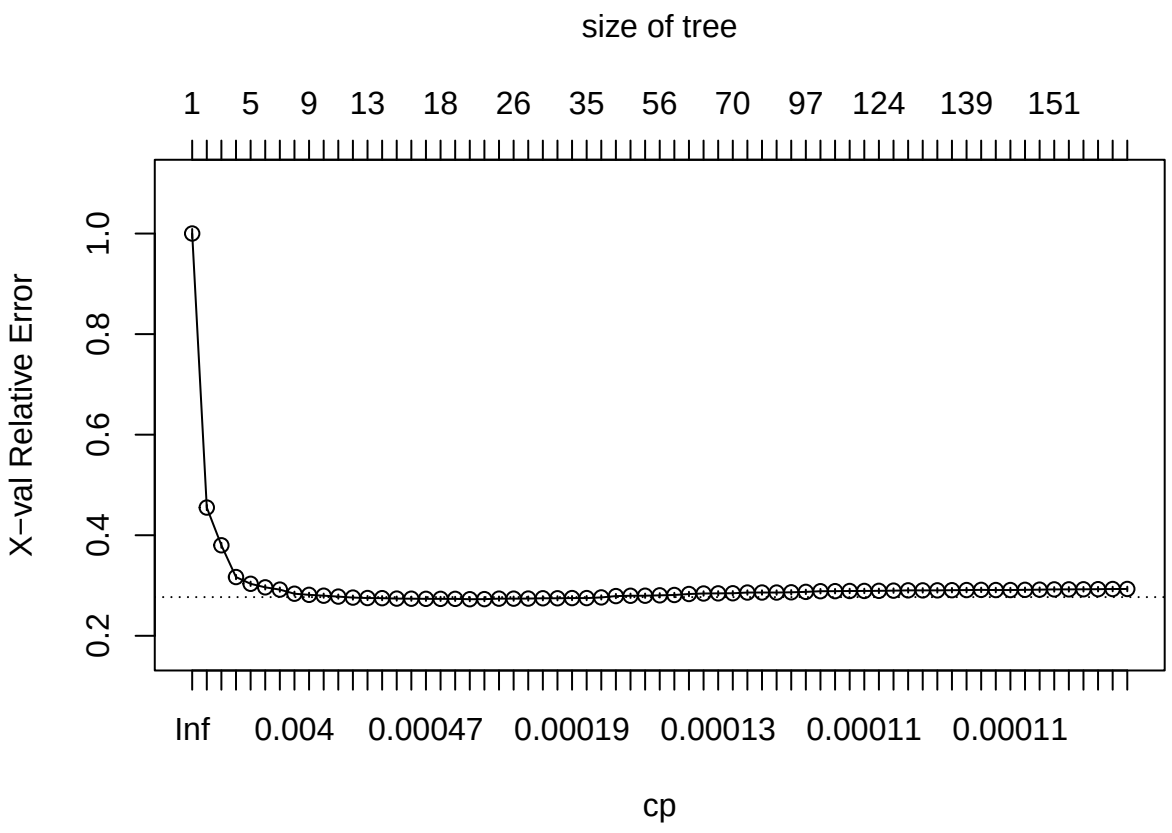

```
# error
kfbest = kknn(price~mileage,train,test,k=kbest,kernel = "rectangular")
sqrt(mean((test$price - kfbest$fitted.values)^2))
```

```
## [1] 9477
```

```
##########
## 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:  185
```

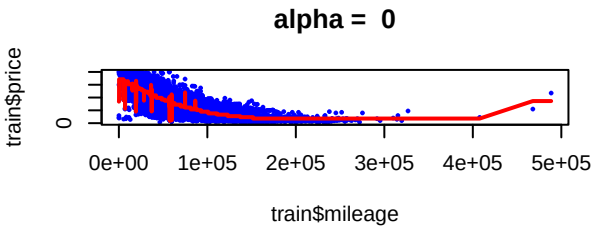
```
plotcp(big.tree) # plot results
```

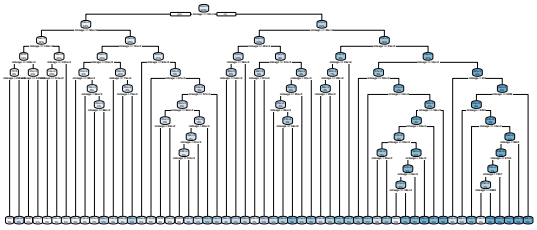


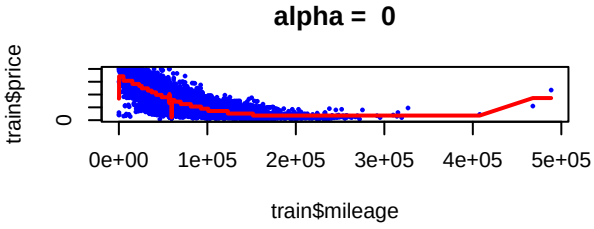

```
cptable = printcp(big.tree)
bestcp = cptable[ which.min(cptable[,"xerror"]), "CP" ]   # this is the optimal cp parameter
```

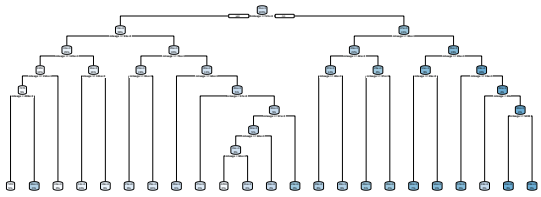
```
# show fit from some trees
oo = order(train$mileage)
cpvec = c(bestcp / 2, bestcp,.0157)

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)
}
```

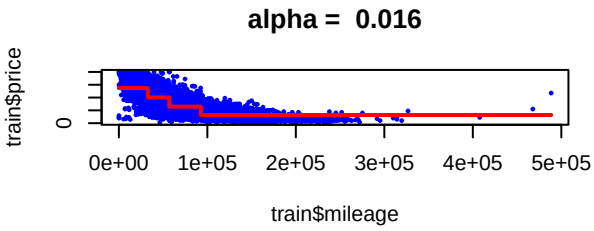
```
## Warning: labs do not fit even at cex 0.15, there
## may be some overplotting
```

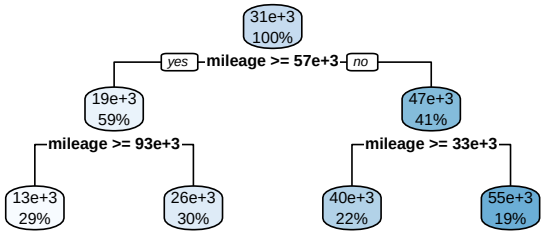




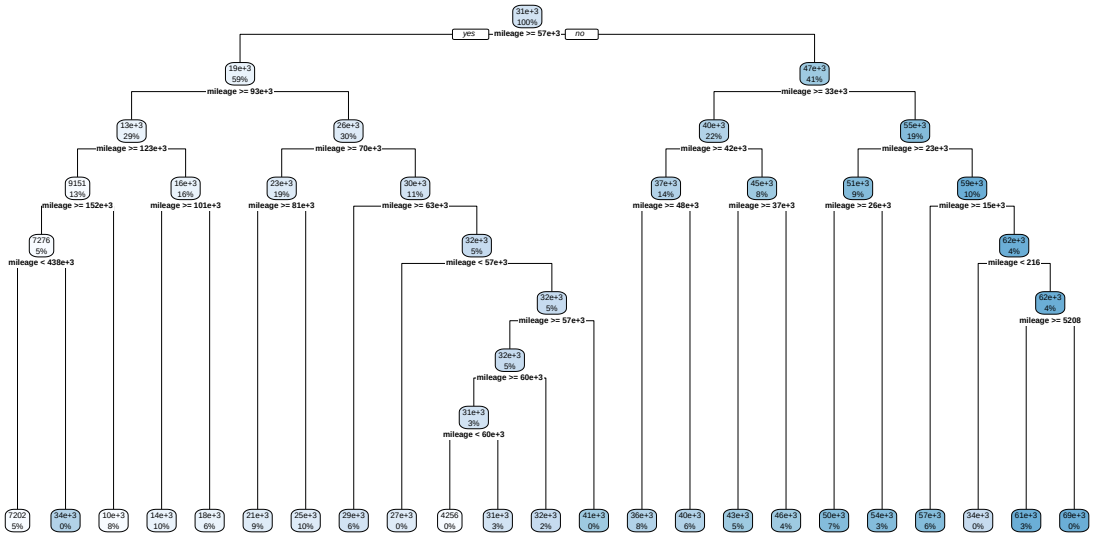




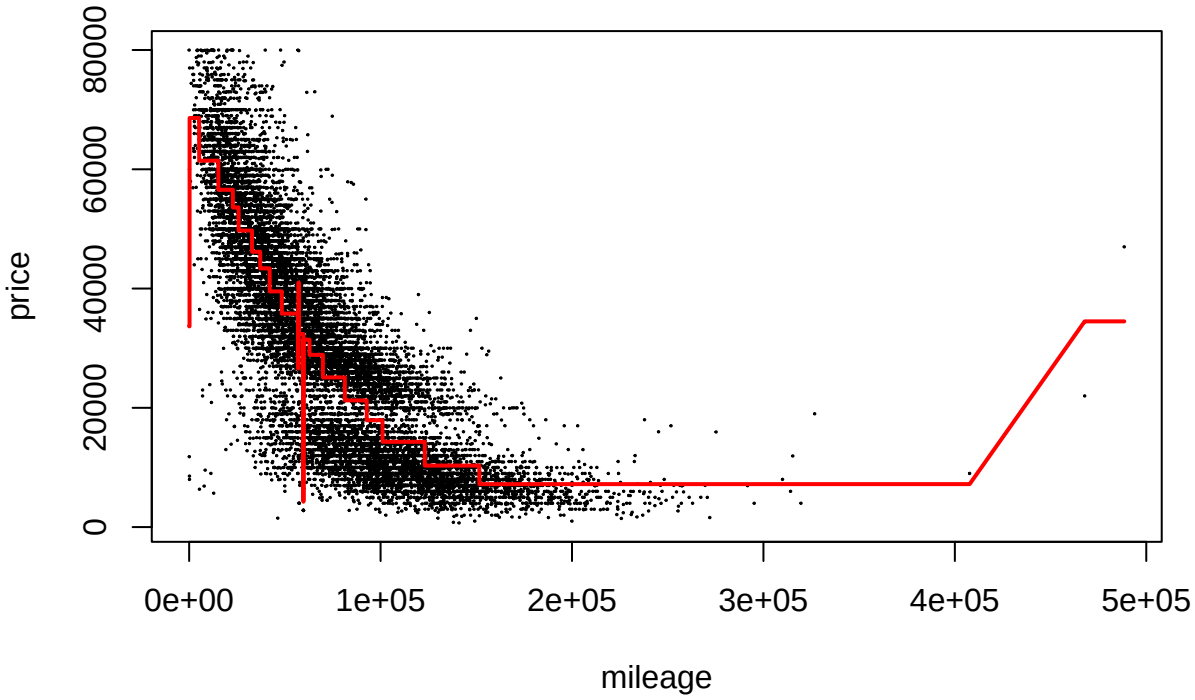





```
par(mfrow=c(1,1))
best.tree = prune(big.tree,cp=bestcp)
rpart.plot(best.tree)
```



```
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] 9596
```

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

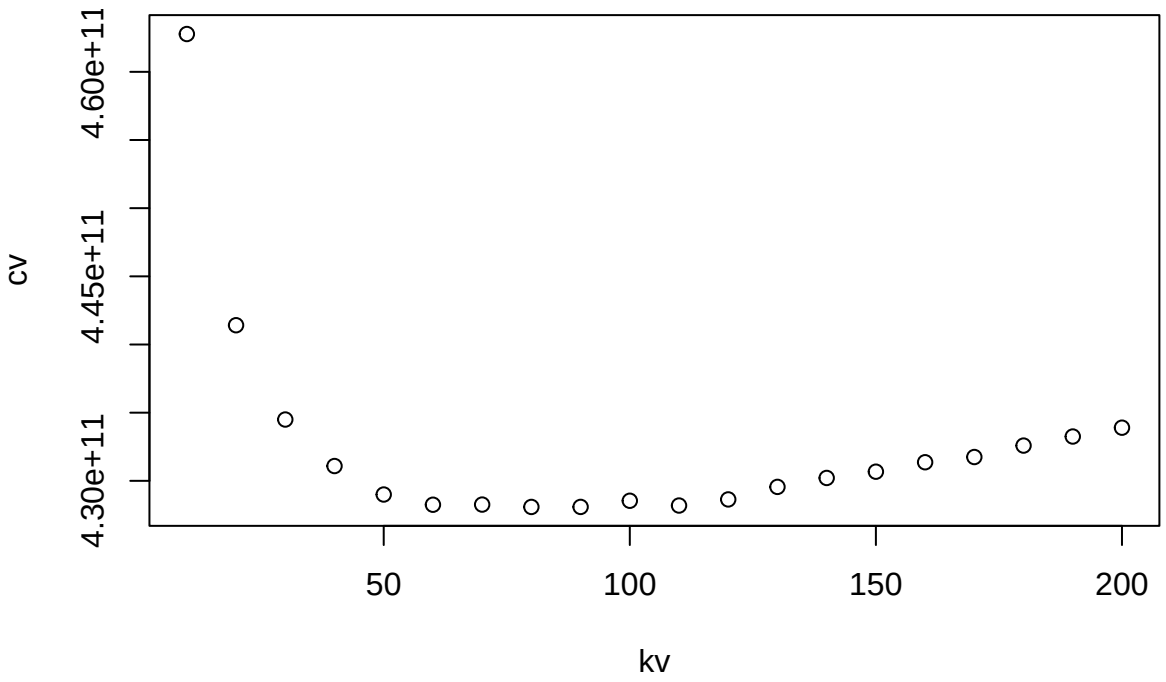
3.4

```
##############################################################################
# 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
```

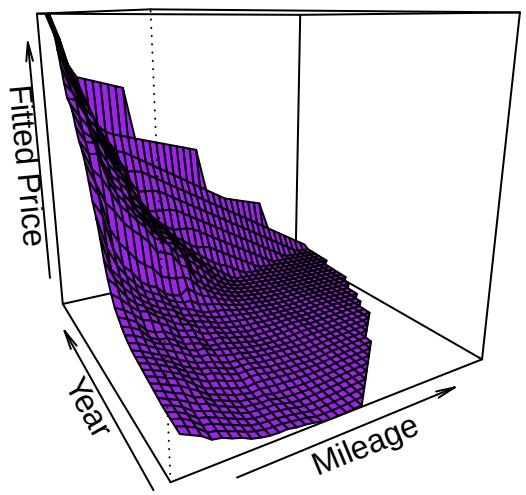
```
plot(kv, cv)
```




```
kbest = kv[which.min(cv)]

# Scaling training set and test set using only the mean and sd from training set
train.scaled = train[order(train$mileage),]
mean.year = mean(train.scaled$year)
sd.year = sd(train.scaled$year)
mean.mileage = mean(train.scaled$mileage)
sd.mileage = sd(train.scaled$mileage)
train.scaled$year = (train.scaled$year - mean.year) / sd.year
train.scaled$mileage = (train.scaled$mileage - mean.mileage) / sd.mileage
test.scaled = test[order(test$mileage),]
test.scaled$year = (test.scaled$year - mean.year) / sd.year
test.scaled$mileage = (test.scaled$mileage - mean.mileage) / sd.mileage

kfbest = kknn(price~mileage+year, train.scaled, train.scaled[,-1], k=kbest,kernel = "rectangular")
```

```
# 3D Perspective plot of fitted values vs mileage and year
library(akima) # package for interpolation
im <- interp(train.scaled$mileage, train.scaled$year, kfbest$fitted.values, duplicate="strip")
persp(x=im$x, y=im$y, z=im$z, xlab="Mileage", ylab="Year", zlab="Fitted Price", theta=-30, main="kNN Fi
```

kNN Fit 3D Perspective Plot



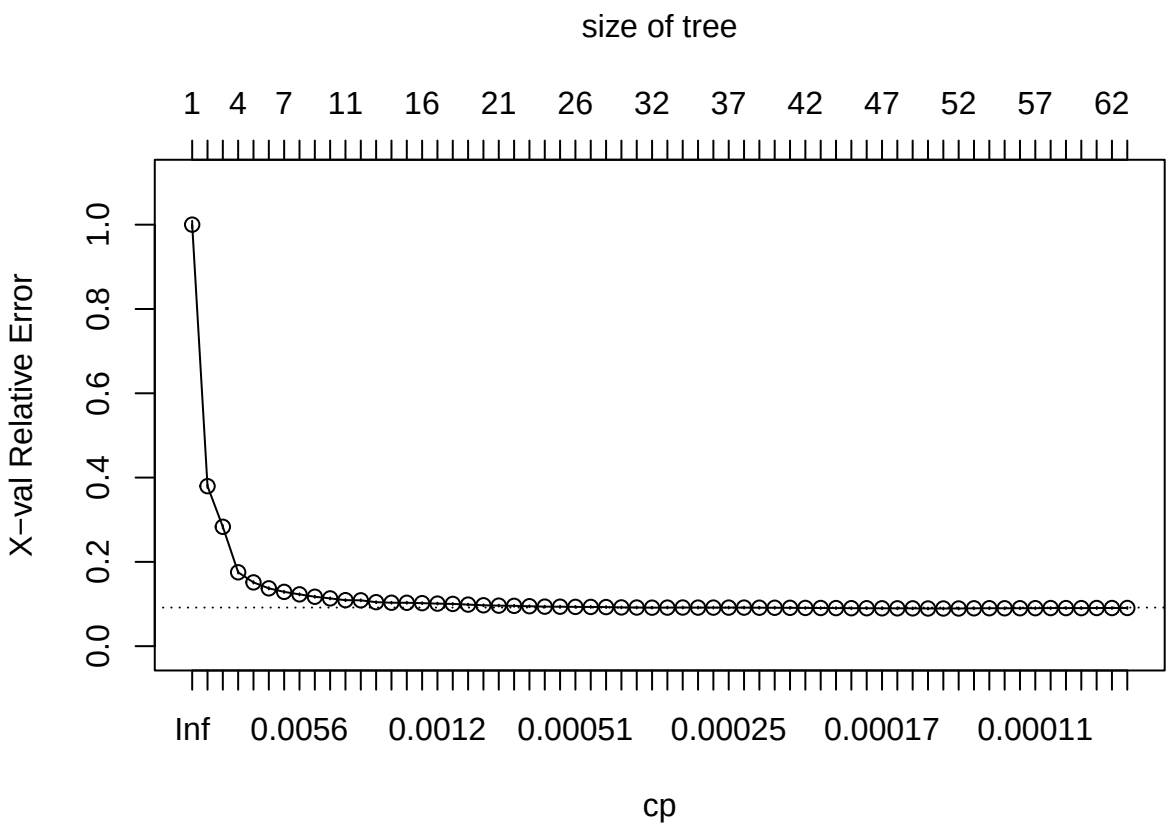

```
# error
kfbest = kknn(price~mileage+year,train.scaled,test.scaled[,-1],k=kbest,kernel = "rectangular")
sqrt(mean((test.scaled$price - kfbest$fitted.values)^2))
```

```
## [1] 5398
```

```
##########
## 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+year, 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')
```

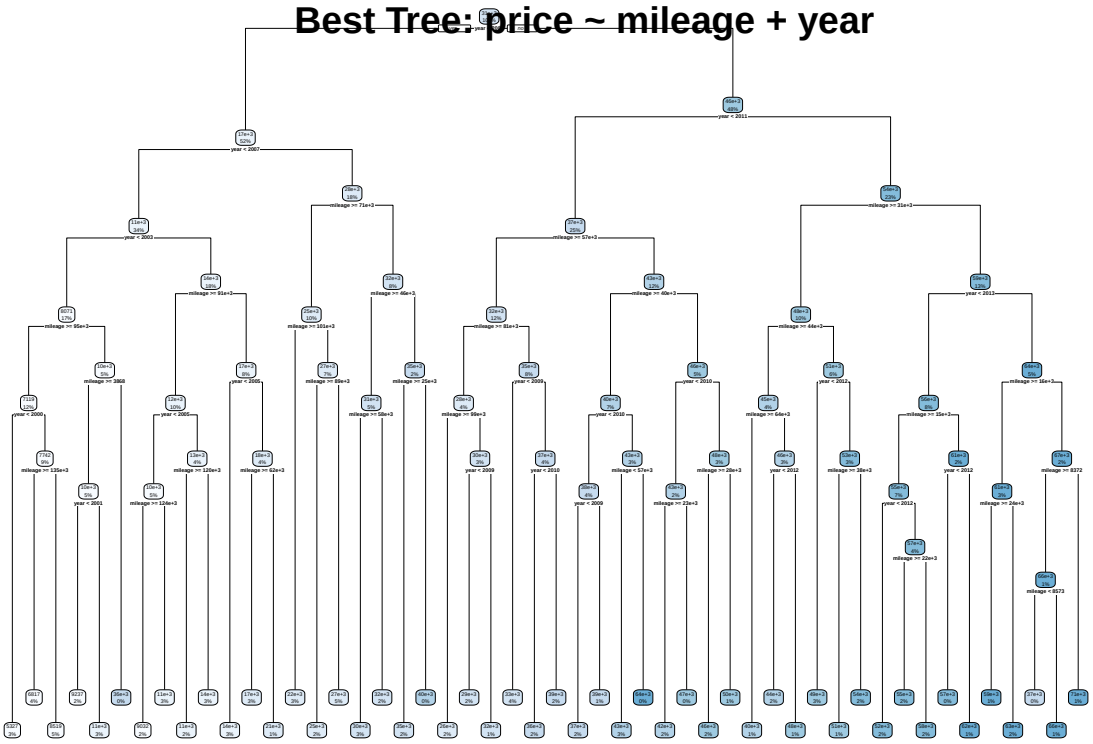
```
## size of big tree:  65
```

```
plotcp(big.tree) # plot results
```

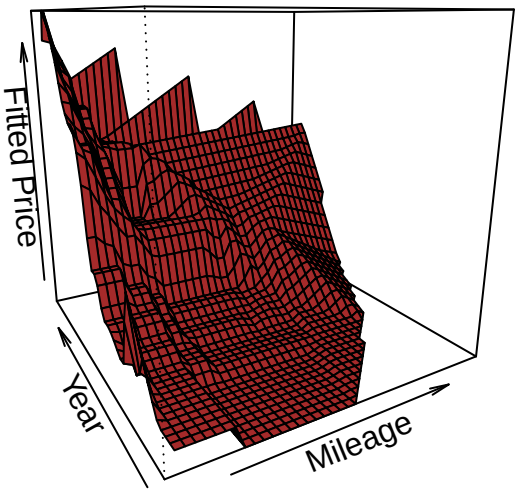



```
cptable = printcp(big.tree)
bestcp = cptable[ which.min(cptable[,"xerror"]), "CP" ]   # this is the optimal cp parameter
```

```
# show fit from some trees
oo = order(train$mileage)

par(mfrow=c(1,1))
best.tree = prune(big.tree,cp=bestcp)
rpart.plot(best.tree)
title("Best Tree: price ~ mileage + year")
```




```
im <- interp(scale(train$mileage)[oo], scale(train$year)[oo], predict(best.tree)[oo], duplicate="strip")
persp(x=im$x, y=im$y, z=im$z, xlab="Mileage", ylab="Year", zlab="Fitted Price", theta=-30, main="Tree Fi
```




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

```
## [1] 5521
```

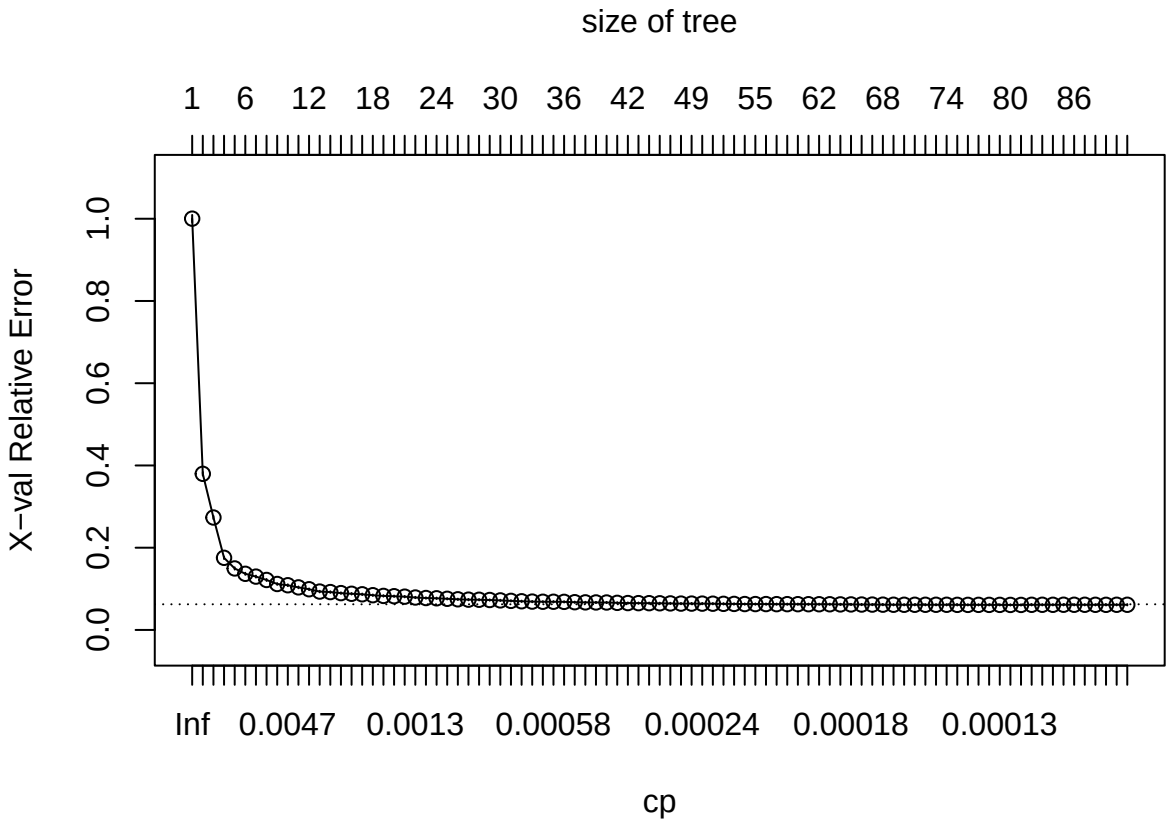
Comparing with one variable case, now k of kNN and optimal tree size are smaller. By introducing the new covariate year, the RMSE of both kNN and regression tree are drastically lower compared to the models using only mileage, i.e. it is performing better.

3.5

```
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:  91
```

```
plotcp(big.tree) # plot results
```

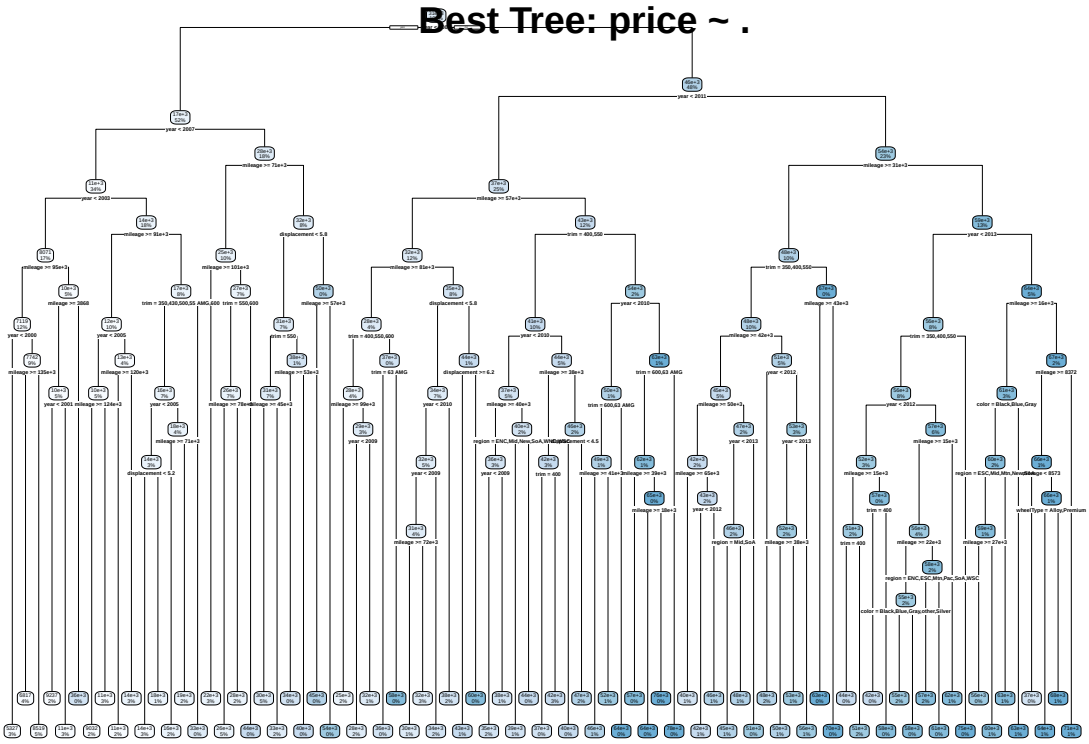



```
cptable = printcp(big.tree)
bestcp = cptable[ which.min(cptable[,"xerror"]), "CP" ]   # this is the optimal cp parameter
```

```
# show fit from some trees
oo = order(train$mileage)

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
```

```
title("Best Tree: price ~ .")
```




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

```
## [1] 4388
```

3.6

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 $k = 1, \ldots, p$. Compare RMSE of all optimal k-variable models and find the one with smallest RMSE.