KNN and B_V tradeoff

Zijian Du

February 2, 2018

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
used_car = read.csv("susedcars.csv")
# get mileage
mileage = used_car$mileage
price = used_car$price
price = c(price)
mileage=c(mileage)
```

plot milage vs price

```
dev.new(width = 6, height = 10)
plot(mileage, price, col="blue", main="mileage-price, linear regression and KNN fit with different Ks")
# run regression, print summary and add line to plot
linear_regression= lm(price~mileage)
print(summary(linear_regression))
```

```
##
## Call:
## lm(formula = price ~ mileage)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -32670  -7063    239   6293  37024
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  5.636e+04  6.706e+02   84.04   <2e-16 ***
## mileage     -3.500e-01  7.870e-03  -44.47   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10670 on 998 degrees of freedom
## Multiple R-squared:  0.6646, Adjusted R-squared:  0.6643
## F-statistic:  1978 on 1 and 998 DF,  p-value: < 2.2e-16
```

```
abline(linear_regression$coef, col="red",lwd=4, xlab="Mileage", ylab="Price")
train=data.frame(mileage, price)
test = data.frame(mileage=sort(mileage))
test$mileage[750]
```

```
## [1] 100042
```

```r
library("kknn")
# fit KNN with various values
k=1
for(k_fold in c(2, 5, 10, 30, 50, 100))
{ color =c("aquamarine","magenta", "cyan", "gold", "maroon", "yellow")
  knn_model = kknn(price~mileage,train,test, k=k_fold, kernel="rectangular")
  lines(test$mileage, knn_model$fitted.values, col = color[k], lwd=0.1)
  k=k+1
}
```

use a good k value = 30 and use linear regression to get the price of car with 100,000 miles on it, which are 16813$ and 21347$

```r
knn_model_optimal = kknn(price~mileage,train,test, k=30, kernel="rectangular")
mileage_sorted= sort(mileage)
print("the 750th element in mileage is 100042")
```

```
## [1] "the 750th element in mileage is 100042"
```

```r
print(mileage_sorted[750])
```

```
## [1] 100042
```

```r
print("predicted value for mileage=100042 with k=30:")
```

```
## [1] "predicted value for mileage=100042 with k=30:"
```

```r
fitted.values(knn_model)[750]
```

```
## [1] 16832.2
```

```r
print("predicted value for mileage=100042 using linear regression:")
```

```
## [1] "predicted value for mileage=100042 using linear regression:"
```

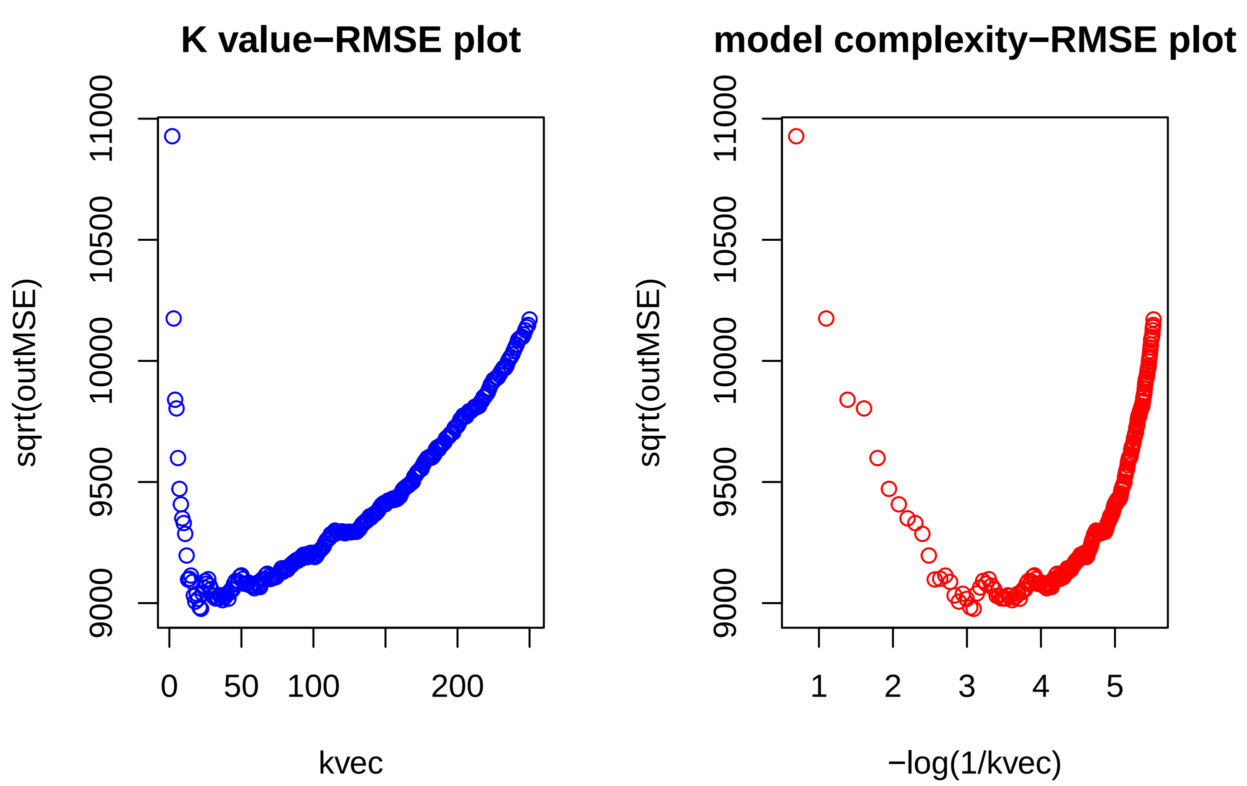
```r
print(predict(linear_regression, test)[750])
```

```
##      750
## 21347.63
```

now we want to plot model complexity-RMSE relationship to see around which value of k will have the lowest RMSE, meaning the optimal point for bias-variance tradeoff, the k value corresponding to smallest RMSE is 21

```r
# create dataframe of milage and price
df=data.frame(mileage, price)
# around 7:3 train test split
ntrain= 700
set.seed(99)
```

```r
tr = sample(1:nrow(df), ntrain)
train = df[tr,]
test = df[-tr,]
kvec=2:250
nk= length(kvec)
outMSE=rep(0,nk)
for (i in 1:nk)
{
  near=kknn(price~mileage, train, test, k=kvec[i], kernel = "rectangular")
  # calculate the MSE as measure
  MSE = mean((test[,2]-near$fitted)^2)
  # put into outMSE vector
  outMSE[i]=MSE
}
#plot
par(mfrow=c(1,2))
plot(kvec,sqrt(outMSE), col = "blue",main="K value-RMSE plot")
plot(-log(1/kvec),sqrt(outMSE), col="red",main ="model complexity-RMSE plot")
```




```r
print(min(sqrt(outMSE)))
```

```
## [1] 8976.332
```

```r
print(which.min(outMSE))
```

```
## [1] 21
```

visualize the bias variance tradeoff for k = 5, 21(optimal), 40 and 300

```r
### simple R function to get fold ids for cross-validation
getfolds = function(nfold,n,dorand=TRUE) { ### set up fold id
## nfold: number of folds (e.g. 5 or 10)
## n: sample size
## dorand: shuffle data
   fs = floor(n/nfold) # fold size
   fid = rep(1:nfold,rep(fs,nfold))
   diff = n-length(fid)
   if(diff>0) fid=c(1:diff,fid)
   if(dorand) fid = sample(fid,n)
   return(fid)
}
```

```r
############################################################
############################################################
## Function to do cross validation.
## docv is a general method that takes a prediction function as an argument.
## docvknn is for kNN, it calls docv handing in a wrapper of kknn.
############################################################
############################################################
#--------------------------------------------------
mse=function(y,yhat) {return(sum((y-yhat)^2))}
doknn=function(x,y,xp,k) {
   kdo=k[1]
   train = data.frame(x,y=y)
   test = data.frame(xp); names(test) = names(train)[1:(ncol(train)-1)]
   near  = kknn(y~.,train,test,k=kdo,kernel='rectangular')
   return(near$fitted)
}
#--------------------------------------------------
docv = function(x,y,set,predfun,loss,nfold=5,doran=TRUE,verbose=TRUE,...)
#x,y training data
#set each row gives settings for predfun
#predfun predicts on xp given (x,y)
#loss: measure of fit
#nfold: number of folds (e.g. 5 or 10)
#doran: should you shuffle the data
{
   #a little error checking
   if(!(is.matrix(x) | is.data.frame(x))) {cat('error in docv: x is not a matrix or data frame\n'); retu
   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)}

   #shuffle the data
   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
```

```
  #loop over folds and settings
  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]
     for(k in 1:nset) { #setting loop
        yhat = predfun(xin,yin,xout,set[k,],...)
        lossv[k]=lossv[k]+loss(yout,yhat)
     }
  }

  return(lossv)
}
#--------------------------------------------------
#cv version for knn
docvknn = function(x,y,k,nfold=5,doran=TRUE,verbose=TRUE) {
return(docv(x,y,matrix(k,ncol=1),doknn,mse,nfold=nfold,doran=doran,verbose=verbose))
}
```

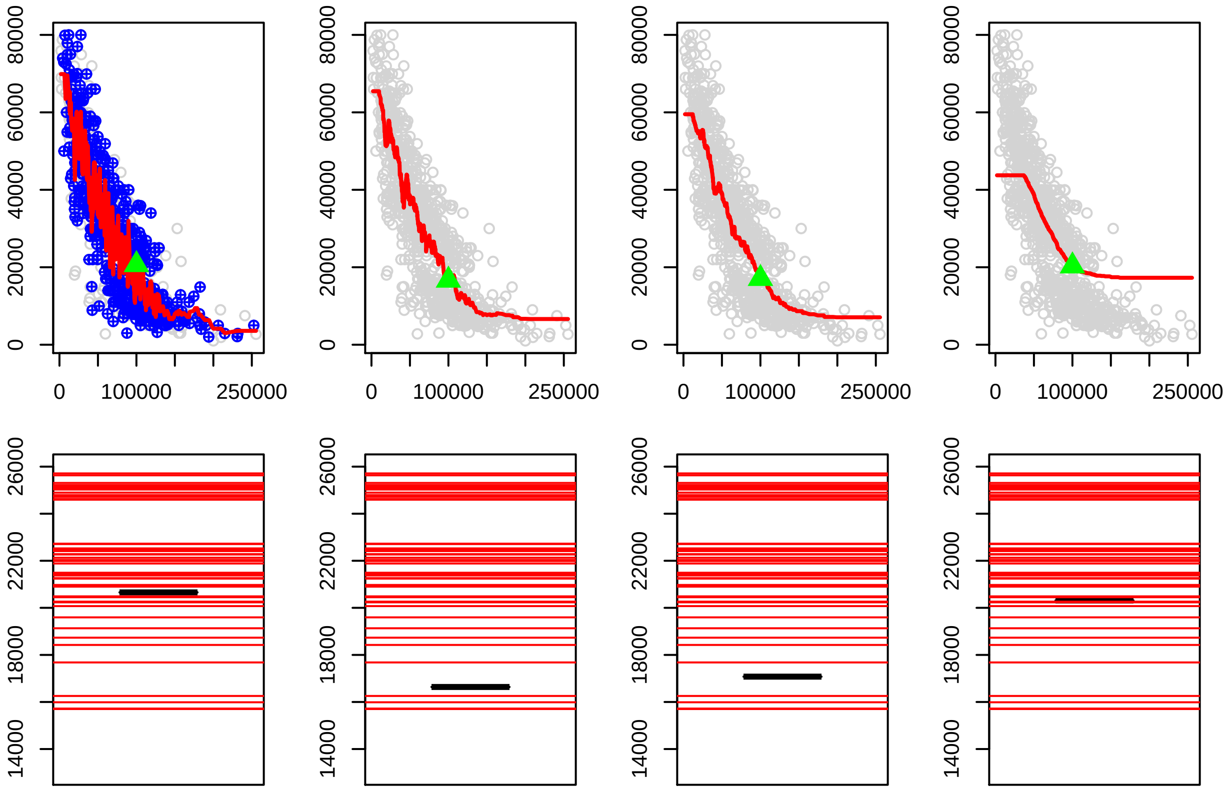
```
######################################################################
if(1) {cat("###load data and libs\n")
#load knn library (need to have installed this with install.packages("kknn"))
library(kknn)
}
```

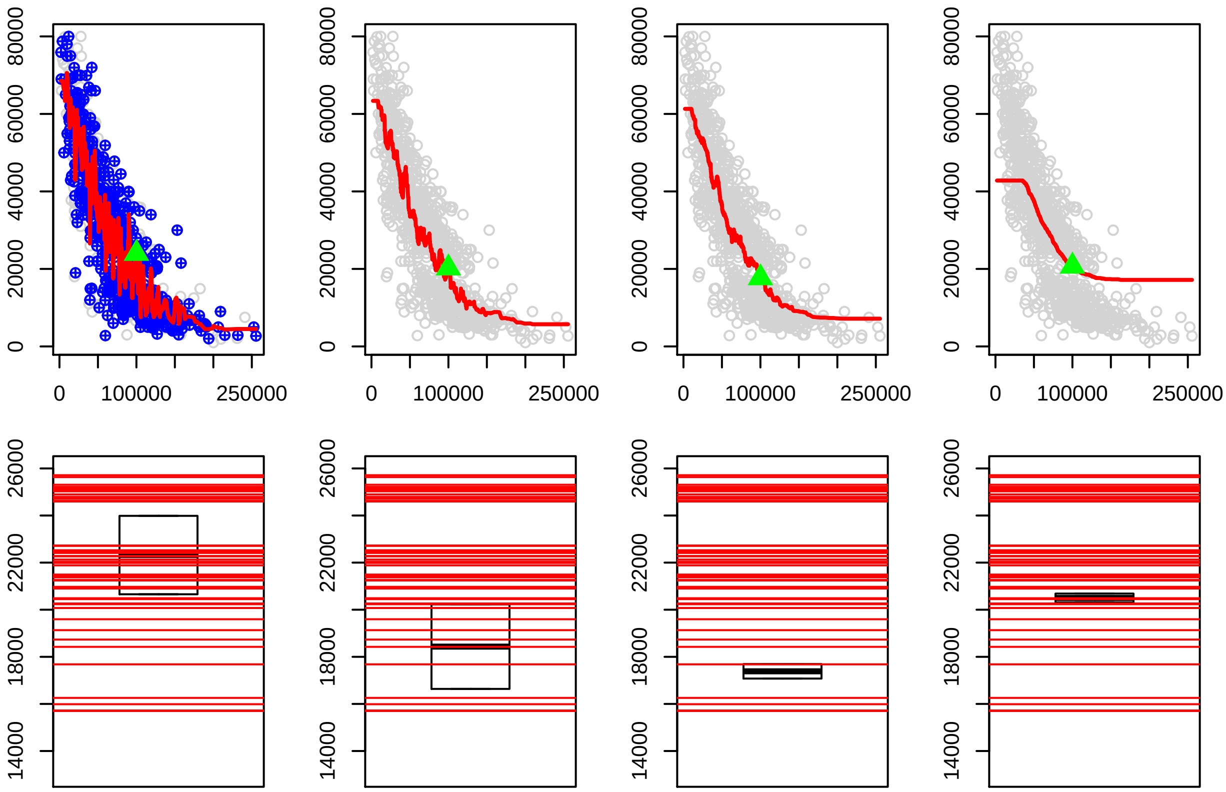
```
## ###load data and libs
```

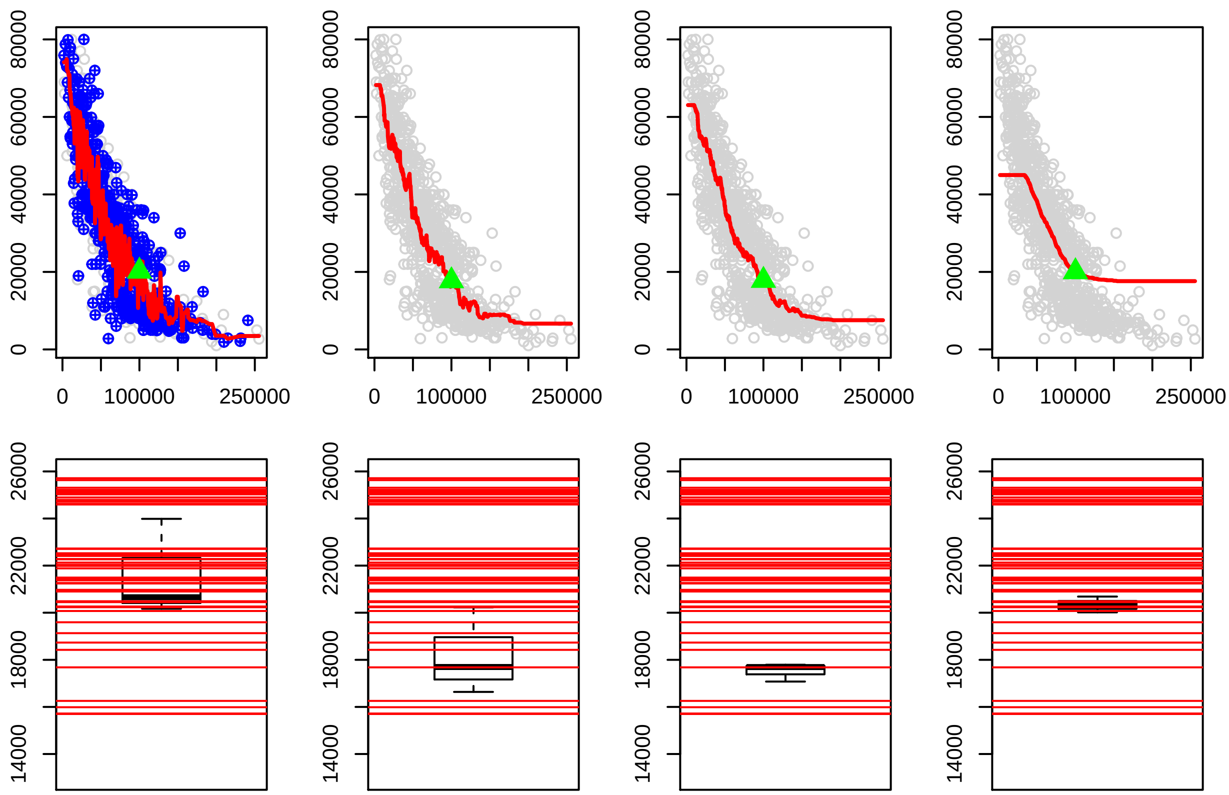
```
######################################################################
if(1) {cat("### run sim\n")
kvec=c(5,21,40,300)
nsim=5
fit1=rep(0,nsim)
fit2=rep(0,nsim)
fit3=rep(0,nsim)
fit4=rep(0,nsim)


train = data.frame(mileage,price)
test = data.frame(mileage=sort(mileage))
par(mfrow=c(2,4))
par("pin"=c(2,2))
par("mar"=c(2,2,2,2))
ntrain=600

fitind = 750
ylm = c(13000,26000)

#get good one
gknn = kknn(mileage~price,train,data.frame(mileage=test[fitind,1]),k=30,kernel = "rectangular")
set.seed(99)
for(i in 1:nsim) {
  ii = sample(1:nrow(train),ntrain)
  kfit1 = kknn(price~mileage,train[ii,],test,k=kvec[1],kernel = "rectangular")
  kfit2 = kknn(price~mileage,train[ii,],test,k=kvec[2],kernel = "rectangular")
```

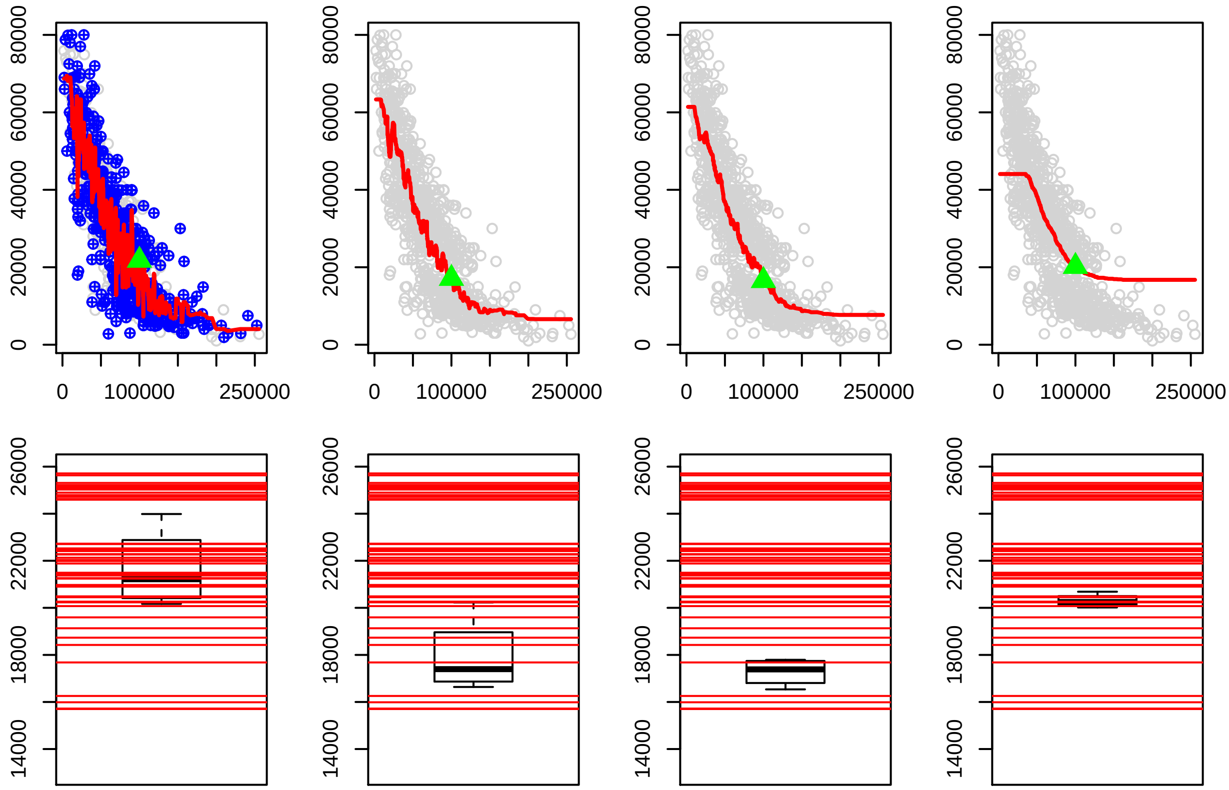
```
  kfit3 = kknn(price~mileage,train[ii,],test,k=kvec[3],kernel = "rectangular")
  kfit4 = kknn(price~mileage,train[ii,],test,k=kvec[4],kernel = "rectangular")
  plot(mileage,price,col="lightgray")
  points(mileage[ii],price[ii],col="blue",pch=10)
  lines(test$mileage,kfit1$fitted,col="red",lwd=2)
  points(test[fitind,1],kfit1$fitted[fitind],col="green",pch=17,cex=2)

  plot(mileage,price,col="lightgray")
  lines(test$mileage,kfit2$fitted,col="red",lwd=2)
  points(test[fitind,1],kfit2$fitted[fitind],col="green",pch=17,cex=2)

  plot(mileage,price,col="lightgray")
  lines(test$mileage,kfit3$fitted,col="red",lwd=2)
  points(test[fitind,1],kfit3$fitted[fitind],col="green",pch=17,cex=2)

  plot(mileage,price,col="lightgray")
  lines(test$mileage,kfit4$fitted,col="red",lwd=2)
  points(test[fitind,1],kfit4$fitted[fitind],col="green",pch=17,cex=2)


  fit1[i]=kfit1$fitted[fitind]
  boxplot(fit1[1:i],ylim=ylm)
  abline(h=gknn$fitted,col="red")

  fit2[i]=kfit2$fitted[fitind]
  boxplot(fit2[1:i],ylim=ylm)
  abline(h=gknn$fitted,col="red")

  fit3[i]=kfit3$fitted[fitind]
  boxplot(fit3[1:i],ylim=ylm)
  abline(h=gknn$fitted,col="red")

  fit4[i]=kfit4$fitted[fitind]
  boxplot(fit4[1:i],ylim=ylm)
  abline(h=gknn$fitted,col="red")

  #readline("go?")
  Sys.sleep(.4)
}
}
```

```
## ### run sim
```

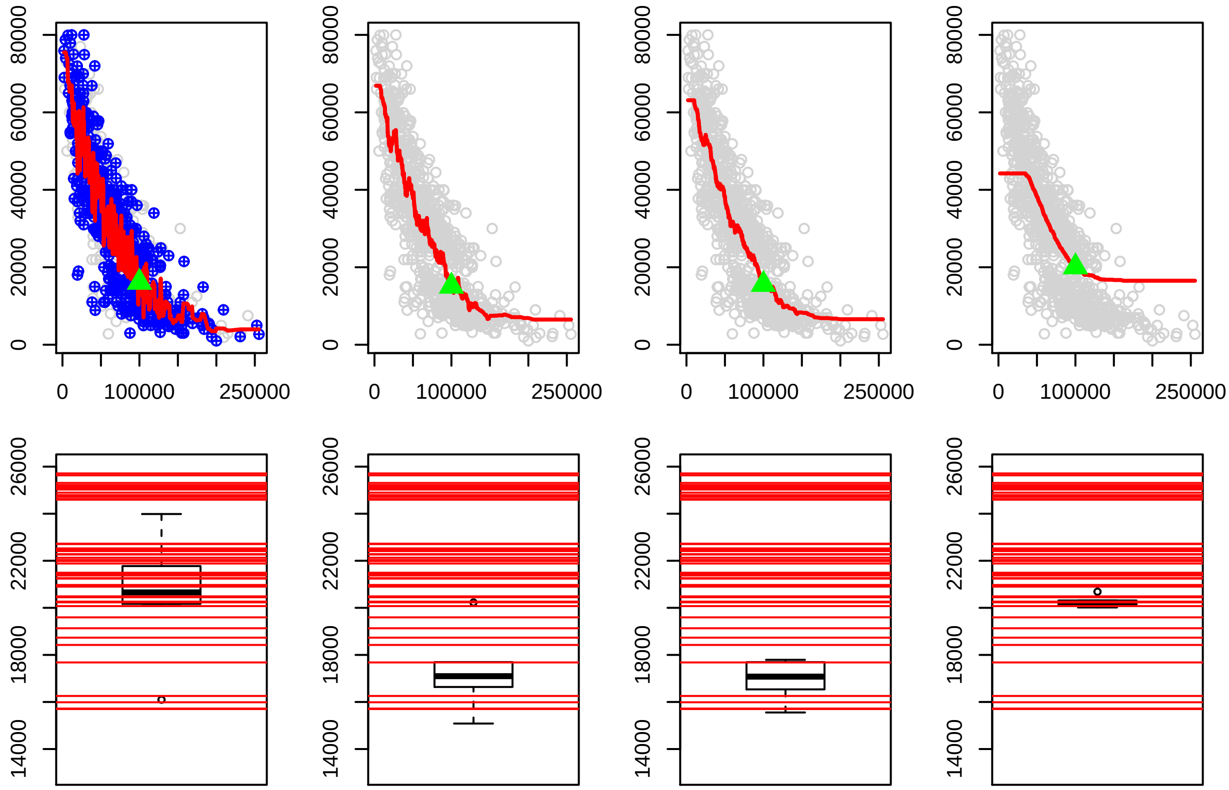

0
20000
40000
60000
80000
0
100000
250000
14000
18000
22000
26000


0
20000
40000
60000
80000
0
100000
250000
14000
18000
22000
26000


0
20000
40000
60000
80000
0
100000
250000
14000
18000
22000
26000



80000
60000
40000
20000
0
0
100000
250000
26000
22000
18000
14000



5 fold cross validation shows that small k results in bigger variance and large k results in bigger bias, which is more obvious when the mileage becomes greater than 150000 miles and the trend price is biased using a large k like 300.

now use all training data to train for k=30 and k=21. Plot the two fits and predict the price for mileage = 100,000 miles using the two models, the predicted prices are 17464$ and 17704$ respectively

```
dev.new(width = 8, height = 10)
plot(mileage, price, col="blue", main="KNN fit with eyeball k=30 and corssvalidation k=21")
train=data.frame(mileage, price)
test = data.frame(mileage=sort(mileage))
test$mileage[750]
```

```
## [1] 100042
```

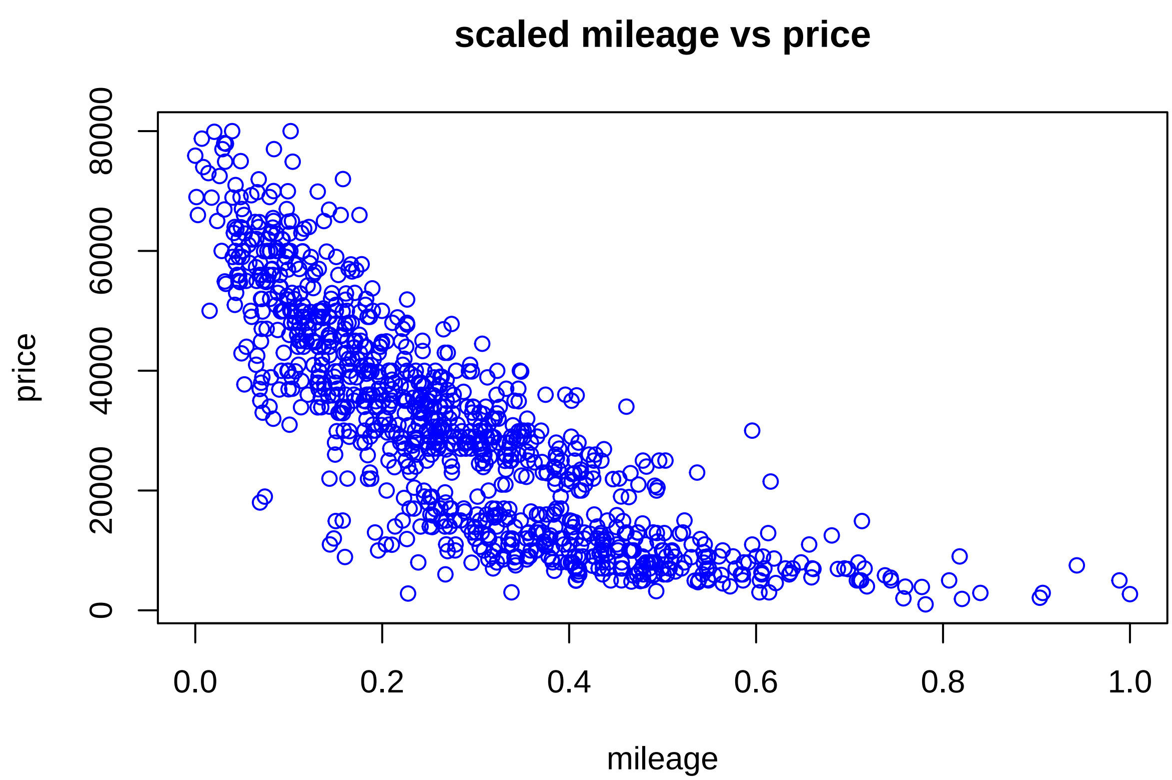
```
library("kknn")
# fit KNN with various values
k=1
fitind=750
for(k_fold in c(30,21))
{ color =c("black", "yellow")
  knn_model = kknn(price~mileage,train,test, k=k_fold, kernel="rectangular")
  lines(test$mileage, knn_model$fitted.values, col = color[k], lwd=2)
```

```r
  points(test[fitind,1],knn_model$fitted[fitind],col=color[k],pch=17,cex=2)
  print(fitted.values(knn_model)[fitind])
  k=k+1
}
```

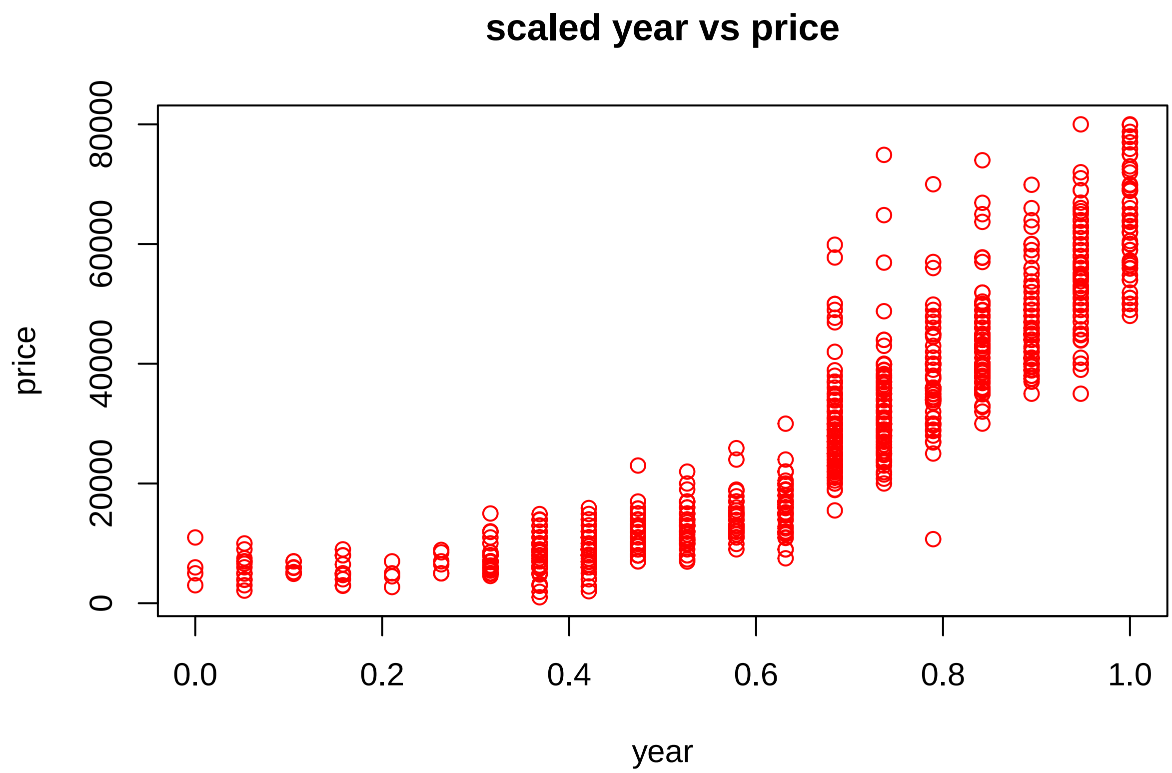
```
## [1] 17464.07
## [1] 18572.67
```

now plot the scaled mileage-price and year-price

```r
# define function to rescale the data
data = cbind(used_car$mileage, used_car$year)
colnames(data)= c("mileage","year")
mmsc = function(x)
{
  return ((x-min(x))/(max(x)-min(x)))
}
scaled_data = apply(data, 2, mmsc)
# plot price vs each x
par(mfrow=c(1,1))
plot(scaled_data[,1], price, col="blue",xlab="mileage", ylab="price", main="scaled mileage vs price")
```




```r
plot(scaled_data[,2], price, col = "red", xlab="year", ylab="price", main="scaled year vs price")
```

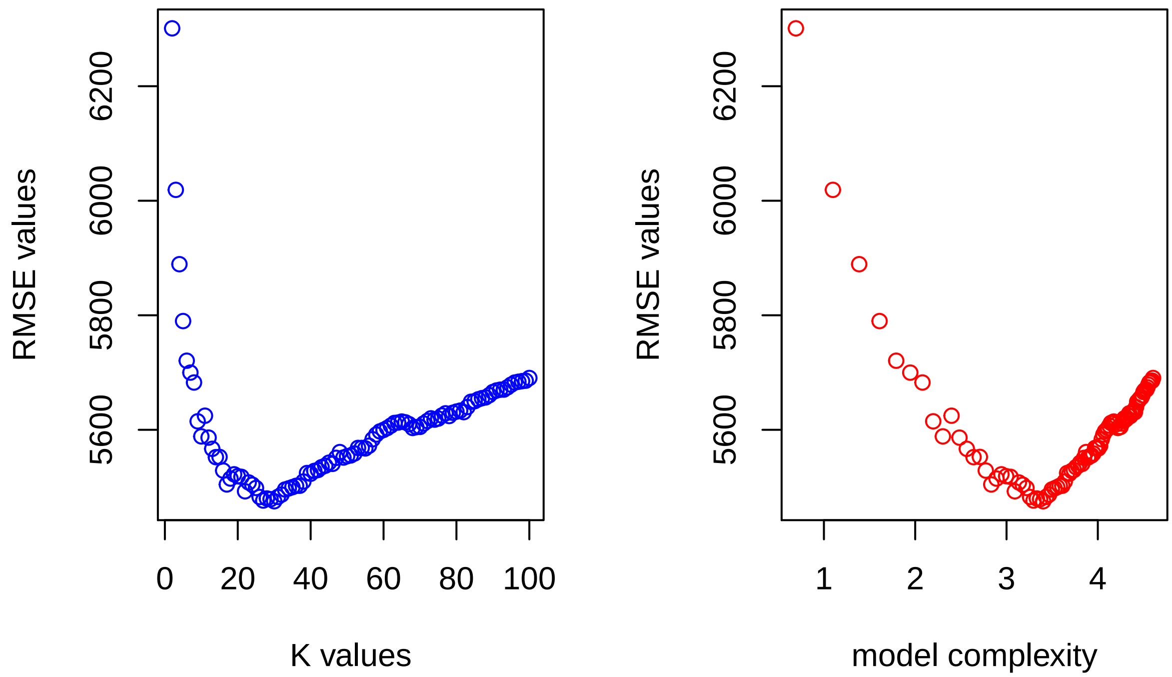


do cross validation and plot the U biase variance tradeoffs

```r
par(mfrow=c(1,2))
set.seed(99)
kv = 2:100
n=length(price)
cvtemp=docvknn(scaled_data, price, kv, nfold=10)
```

```
## in docv: nset,n,nfold:  99 1000 10
## on fold:  1 , range:  1 : 100
## on fold:  2 , range:  101 : 200
## on fold:  3 , range:  201 : 300
## on fold:  4 , range:  301 : 400
## on fold:  5 , range:  401 : 500
## on fold:  6 , range:  501 : 600
## on fold:  7 , range:  601 : 700
## on fold:  8 , range:  701 : 800
## on fold:  9 , range:  801 : 900
## on fold:  10 , range:  901 : 1000
```

```r
cvtemp =sqrt(cvtemp/n)
plot(kv, cvtemp, col = "blue", xlab="K values", ylab="RMSE values")
plot(-log(1/kv), cvtemp, col="red", xlab="model complexity", ylab="RMSE values")
```




```
print(min(cvtemp))
```

```
## [1] 5475.27
```

```
print(which.min(cvtemp))
```

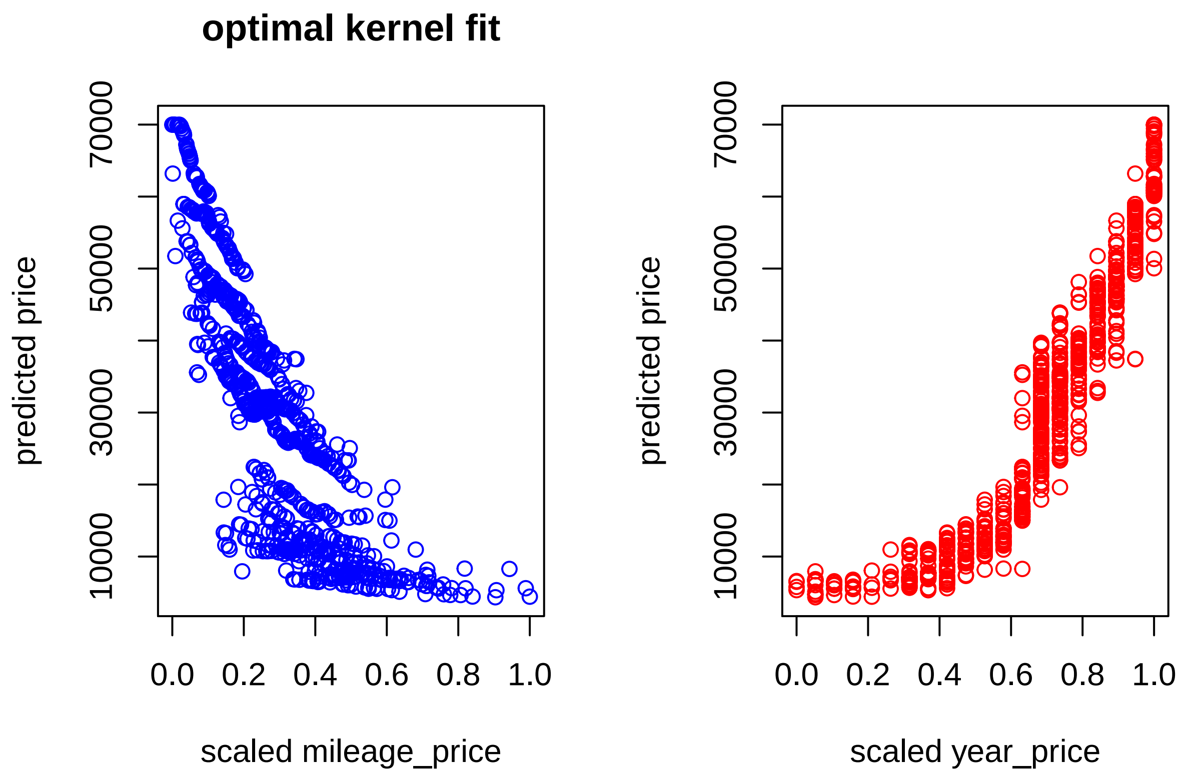
```
## [1] 29
```

using two features indeed increased the accuracy, the minimum RMSE decrease from 8976 to 5475 when two features are used. The optimal K value increase from 21 to 29 using two features. This means that by using two features and taking into consideration more neighbours the prediction accuracy can imporve.

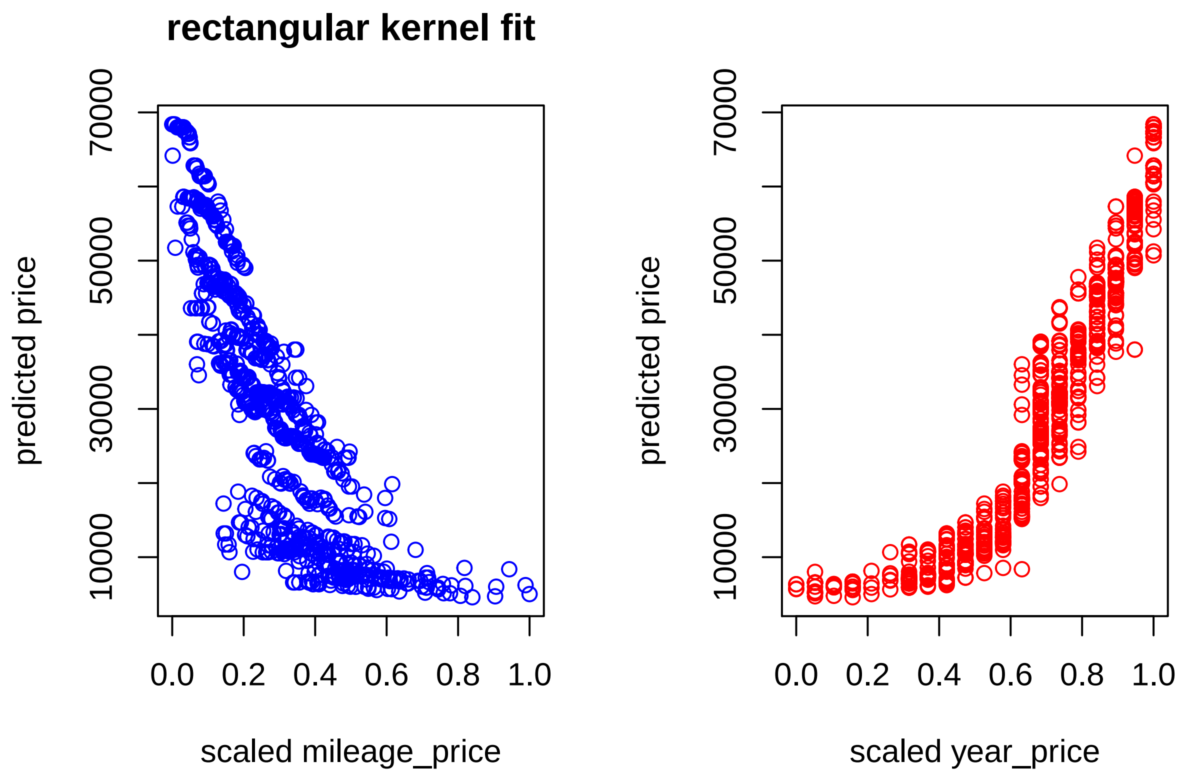
now experiment with the choice of kernels, use the two features scaled data and

```
# create dataframe of milage and price
library("kknn")
knn_optimal =kknn(price~scaled_data, train, test,k=29, kernel="optimal")
knn_rec =kknn(price~scaled_data, train, test, k=29,kernel="rectangular")
par(mfrow=c(1,2))
```

```
plot(scaled_data[,1],fitted.values(knn_optimal), col="blue", xlab="scaled mileage_price", ylab = "predi
plot(scaled_data[,2],fitted.values(knn_optimal), col ="red", xlab="scaled year_price", ylab="predicted p
```




```
par(mfrow=c(1,2))
plot(scaled_data[,1],fitted.values(knn_rec), col="blue", xlab="scaled mileage_price", ylab = "predicted
plot(scaled_data[,2],fitted.values(knn_rec), col ="red", xlab="scaled year_price", ylab="predicted pric
```




```r
MSE_optimal = mean((test[,1]-knn_optimal$fitted)^2)
MSE_rectangular = mean((test[,1]-knn_rec$fitted)^2)
print(MSE_optimal)
```

```
## [1] 4004131506
```

```r
print(MSE_rectangular)
```

```
## [1] 3996465866
```

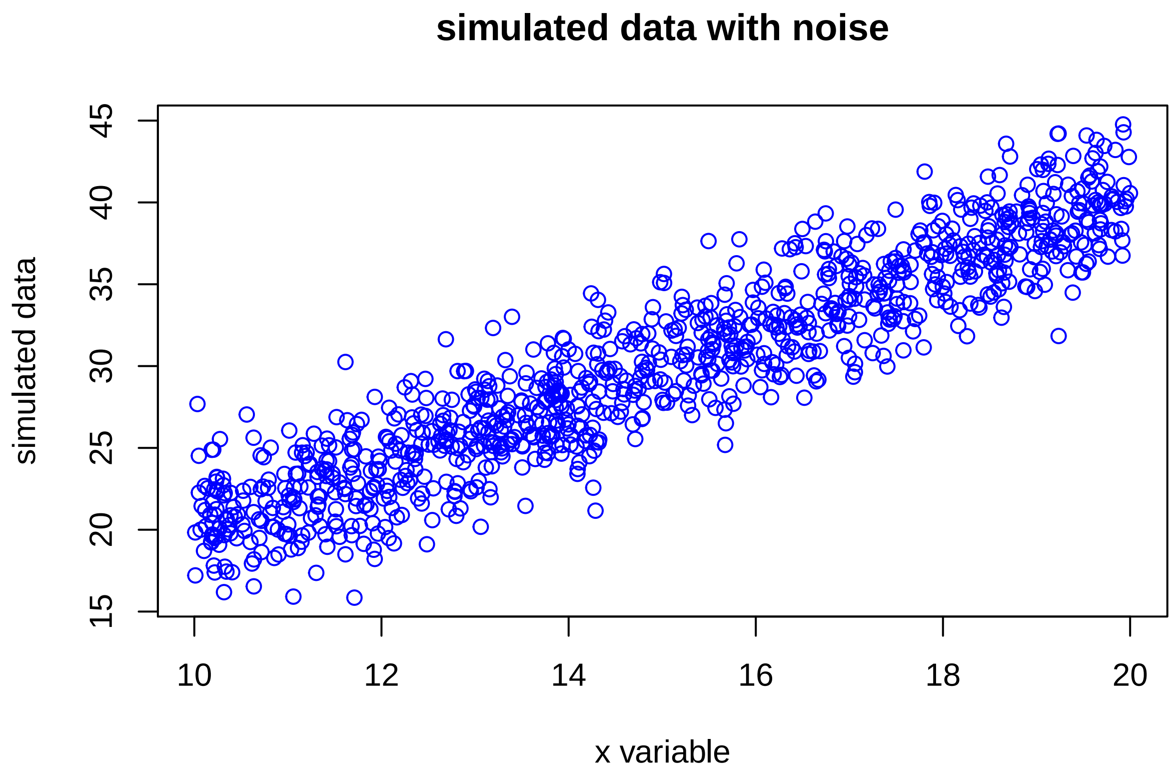
it turned out that optimal kernel has slightly more MSE value compared to using rectangular kernel.

illustrate bias variance tradeoff with simulation

```r
library(simstudy)
```

```
## Loading required package: data.table
```

```r
def <- defData(varname = "nr", dist = "nonrandom", formula = 0, id = "idnum")
def <- defData(def, varname = "x1", dist = "uniform", formula = "10;20")
def <- defData(def, varname = "y1", formula = "nr + x1 * 2", variance = 6)
dt <- genData(1000,def)
plot(dt$x1, dt$y1, col="blue", xlab="x variable",ylab="simulated data", main="simulated data with noise
```

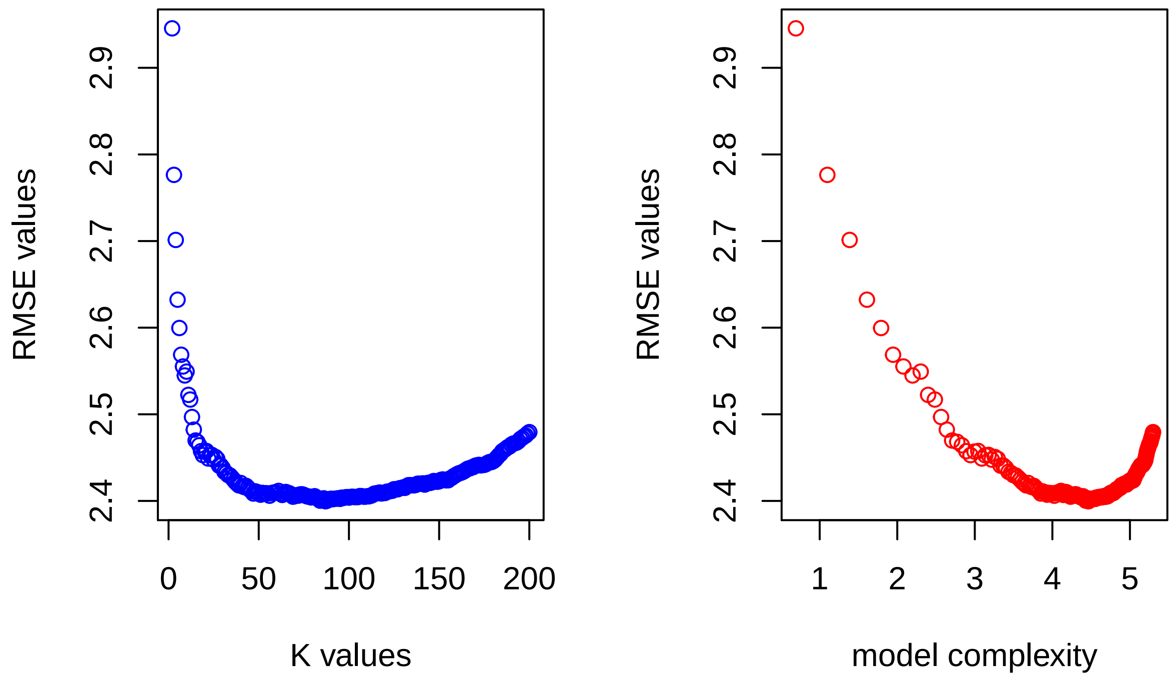


bias-variance tradeoff for the simulated data

```
par(mfrow=c(1,2))
set.seed(99)
kv = 2:200
n=length(price)
cvtemp=docvknn(data.frame(dt$x1),c(dt$y1), kv, nfold=5)
```

```
## in docv: nset,n,nfold:  199 1000 5
## on fold:  1 , range:  1 : 200
## on fold:  2 , range:  201 : 400
## on fold:  3 , range:  401 : 600
## on fold:  4 , range:  601 : 800
## on fold:  5 , range:  801 : 1000
```

```
cvtemp =sqrt(cvtemp/n)
plot(kv, cvtemp, col = "blue", xlab="K values", ylab="RMSE values")
plot(-log(1/kv), cvtemp, col="red", xlab="model complexity", ylab="RMSE values")
```




```r
print(min(cvtemp))
```

```
## [1] 2.399632
```

```r
print(which.min(cvtemp))
```

```
## [1] 86
```