Model assessment

We'll now look at different model assessment techniques, focusing on variance/bias of their estimate and computational cost.

We'll generate data from a linear model (with white noise) and use KNN to estimate the curve.

```
# Dataset generation
f1dim<-function(x){ sin(8*x)/(1+(4*x)^2) }

DataGenerator <- function(n, p, sd.x, sd.eps) {
  X <- replicate(p, rnorm(n, sd = sd.x))
  eps <- rnorm(n, sd=sd.eps)
  Y <- f1dim(X[,1]) + eps
  return(data.frame(Y = Y, X = X))
}
```

```
library(kknn)
```

```
## Warning: package 'kknn' was built under R version 3.6.3
```

```
train <- DataGenerator(n=200, p=10, sd.x=5, sd.eps=1)
test <- DataGenerator(n=100, p=10, sd.x=5, sd.eps=1)
knn.8 <- kknn(formula=Y~. , train = train, test = test, k = 8)
```

```
head(test)
```

```
##            Y         X.1          X.2         X.3        X.4         X.5
## 1  0.9103076  0.06942449  2.360602443  -0.5081367  8.2101244  -2.870211
## 2 -2.0638792 -1.42093718 -0.898021087  -1.2955147  2.3476158  -5.499377
## 3 -0.4045688  2.74867494  3.538634722 -10.0169459  1.1885799   3.777089
## 4  1.0723297  7.38177443 -3.857601588 -11.3296315  0.4596525   7.059043
## 5  1.7899504 -2.37381948  3.513945773   0.2728810 -4.7464513 -12.617067
## 6 -1.2627489 -4.79510351  0.004338688  -4.8477015  2.1551797   1.280222
##         X.6        X.7        X.8       X.9       X.10
## 1  3.549131 -0.5063548  0.5690146  3.503210 -8.2317666
## 2  3.475010  6.1431585  3.8123488  6.448057 -6.2461192
## 3 -6.886993 -5.0397801  7.1931498 -2.777438 -2.3817457
## 4  5.738708 -2.6124240  0.3222313  2.396278  4.9028681
## 5 -2.434192 -1.1168876  7.0706788  7.760494 -0.7916563
## 6 -6.016205 -1.6804068 -2.6856883  3.273530 -7.8407198
```

```
test.preds <- predict(knn.8)
MSE.test <- sum((test$Y - test.preds)**2)/100
MSE.test
```

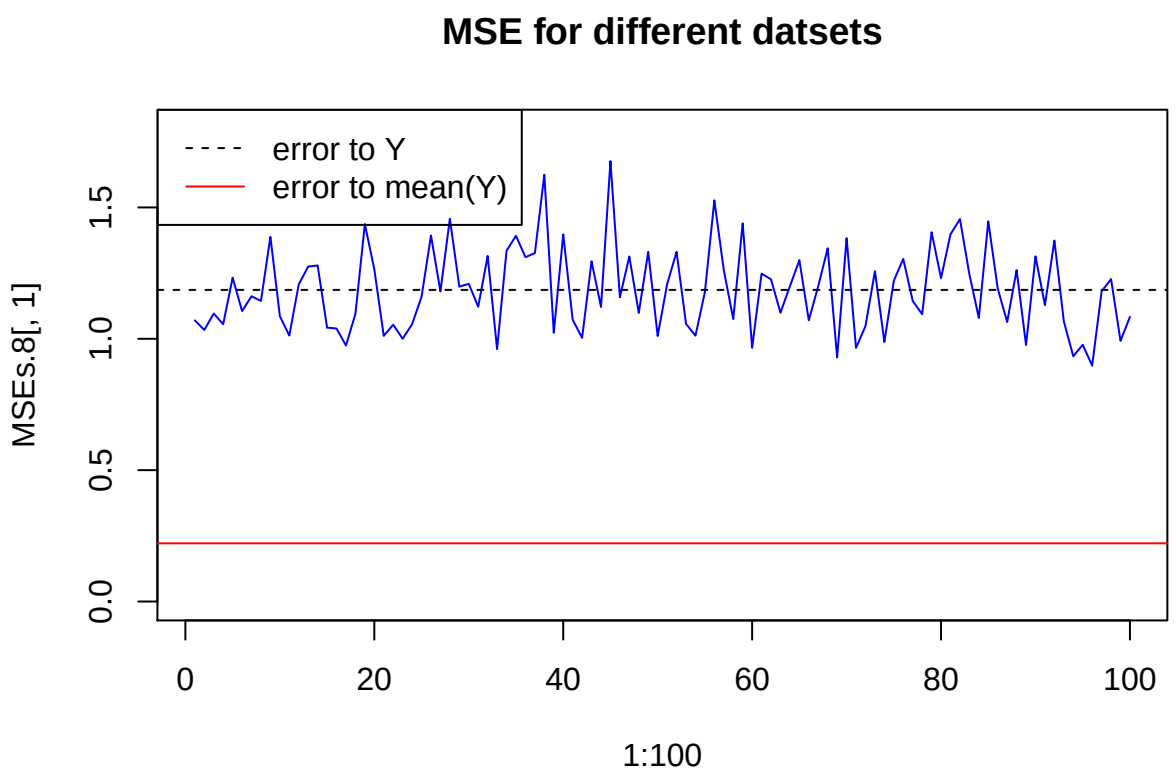
```
## [1] 1.738999
```

Now we'll simulate 100 datasets to approximate the test MSE.

```
nsim <- 100
simulate.knn <- function(nsim, k, sd.x=5, sd.eps=1){
  MSEs <- matrix(nrow=nsim, ncol=2)
```

```
  for(i in 1:nsim){
    train <- DataGenerator(n=200, p=10, sd.x=sd.x, sd.eps=sd.eps)
    test <- DataGenerator(n=100, p=10, sd.x=sd.x, sd.eps=sd.eps)
    knn.8 <- kknn(formula=Y~. , train = train, test = test, k = k)
    test.preds <- predict(knn.8)
    MSE.test <- sum((test$Y - test.preds)**2)/100
    MSEs[i,1]<-MSE.test
    MSE.true <- (sum((f1dim(test$X.1)-test.preds)**2))/100
    MSEs[i,2]<-MSE.true
  }
  return(MSEs)
}
```

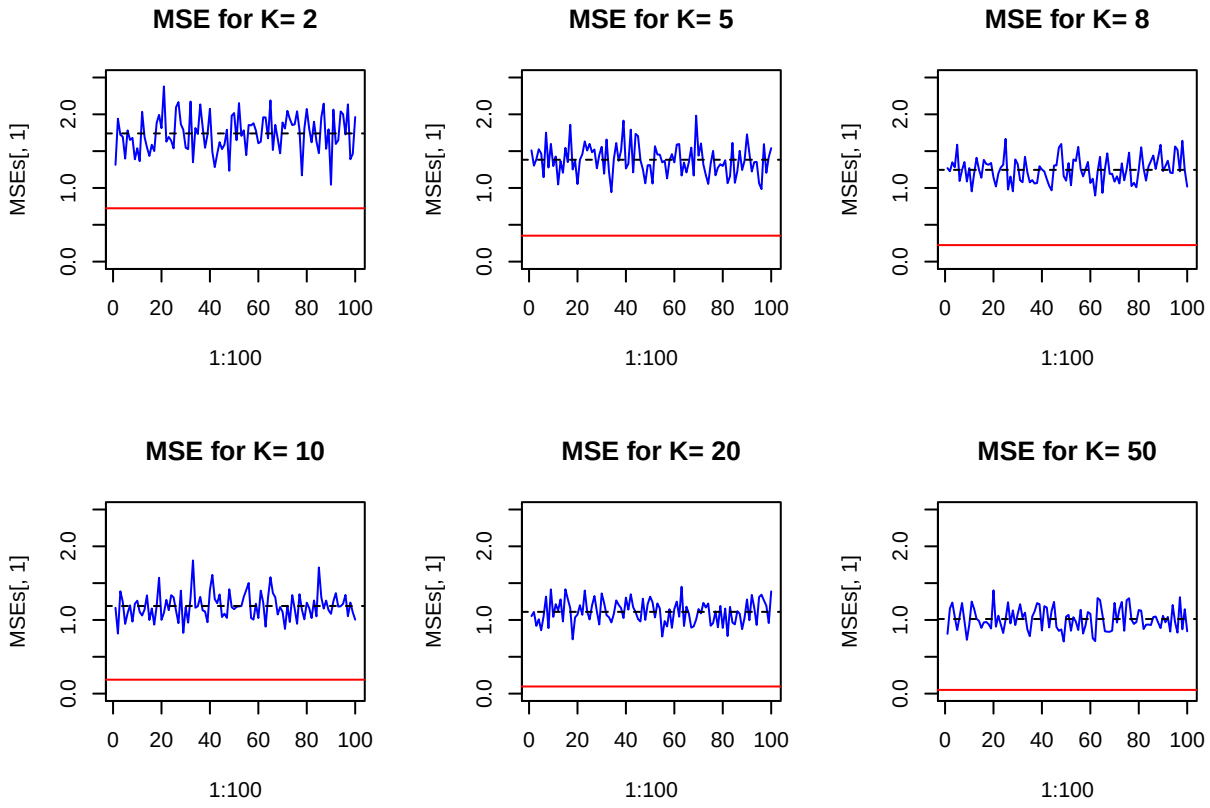
```
MSEs.8 <- simulate.knn(100, 8)
plot(1:100, MSEs.8[,1], type="l", main="MSE for different datsets", col="blue", ylim = c(0,1.8))
abline(h=mean(MSEs.8[,1]), lty="dashed")
abline(h=mean(MSEs.8[,2]), col="red")
legend("topleft", legend = c("error to Y","error to mean(Y)"), col = c("black","red"),lty =c("dashed","
```



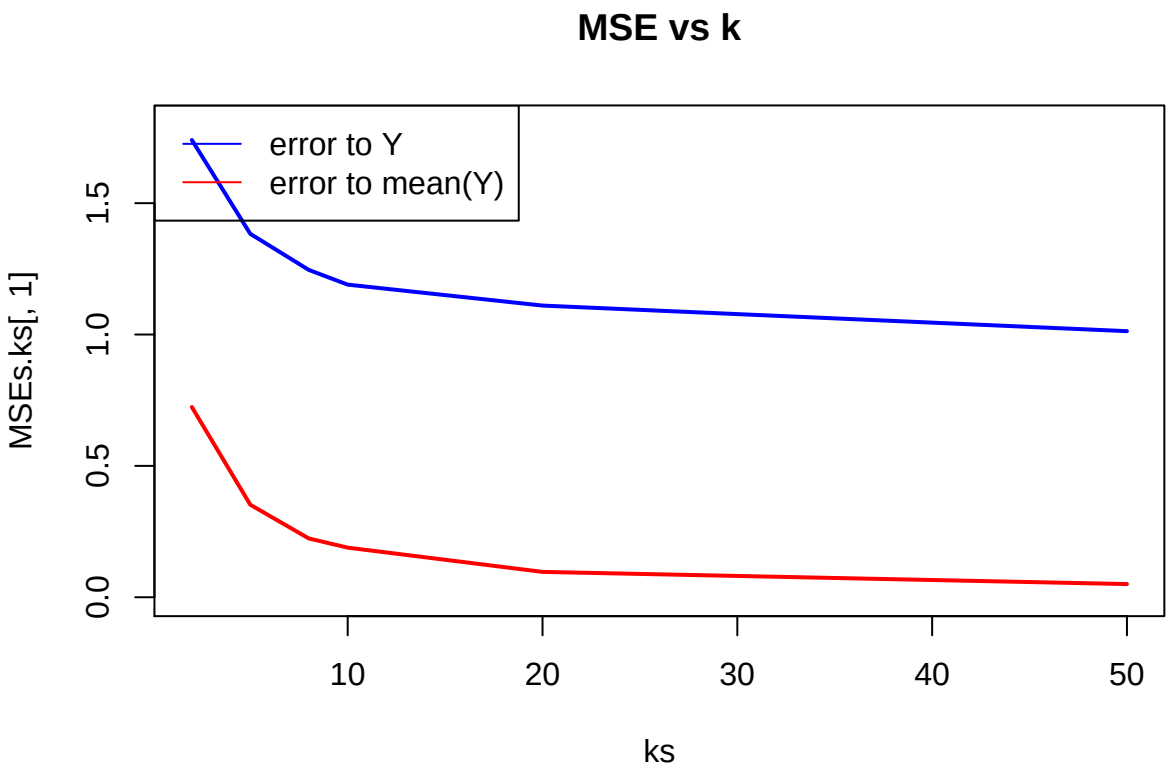
Now let's investigate how the MSE would change for different k.

```
ks <- c(2,5,8,10,20,50)
par(mfrow=c(2,3))
MSEs.ks <- matrix(nrow = 6, ncol=2)
for(i in 1:length(ks)){
  k <- ks[i]
  MSEs <- simulate.knn(100, k)
```

```
  plot(1:100, MSEs[,1], type="l", main=paste("MSE for K=",k), col="blue", ylim = c(0,2.5))
  abline(h=mean(MSEs[,1]), lty="dashed")
  abline(h=mean(MSEs[,2]), col="red")
  MSEs.ks[i,1] <- mean(MSEs[,1])
  MSEs.ks[i,2] <- mean(MSEs[,2])
}
```



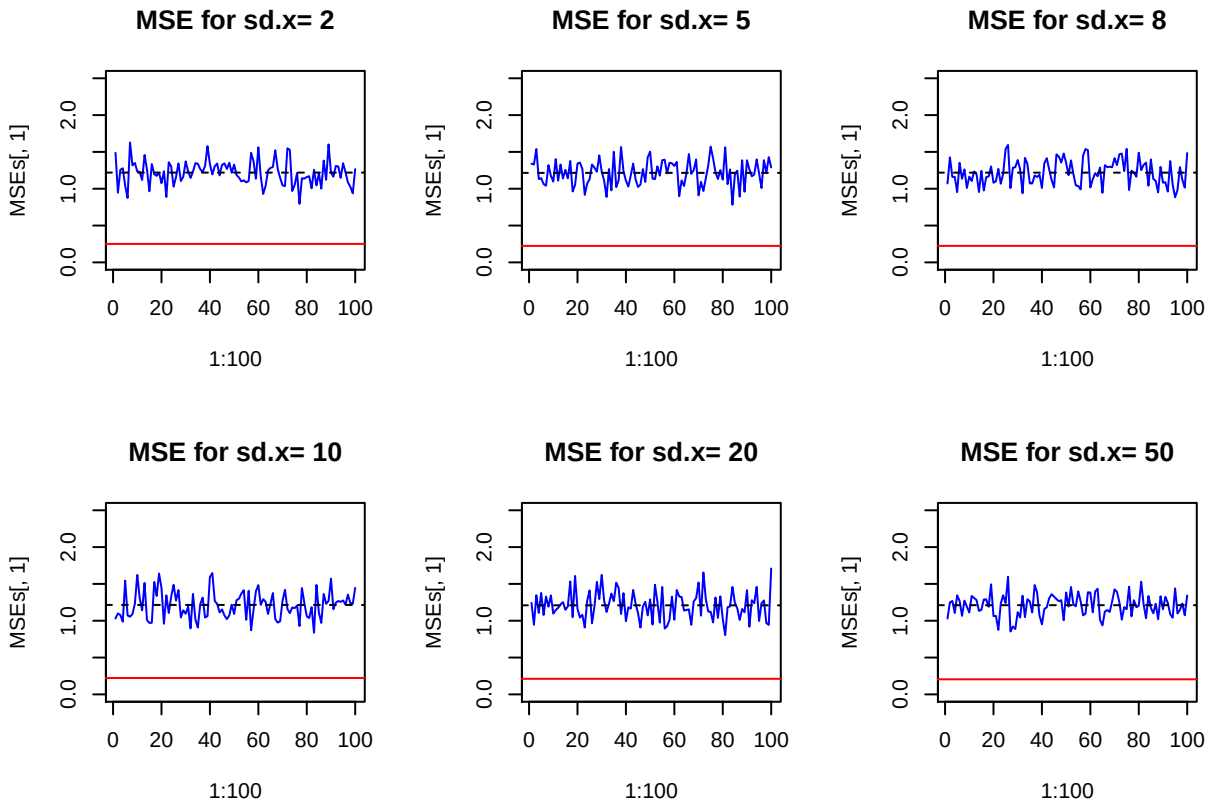

```
par(mfrow=c(1,1))
plot(ks, MSEs.ks[,1], type="l", main = "MSE vs k", col="blue", lwd=2, ylim=c(0,1.8))
points(ks, MSEs.ks[,2], type="l", col="red", lwd=2)
legend("topleft", legend = c("error to Y","error to mean(Y)"), col = c("blue","red"),lty =c("solid","so
```



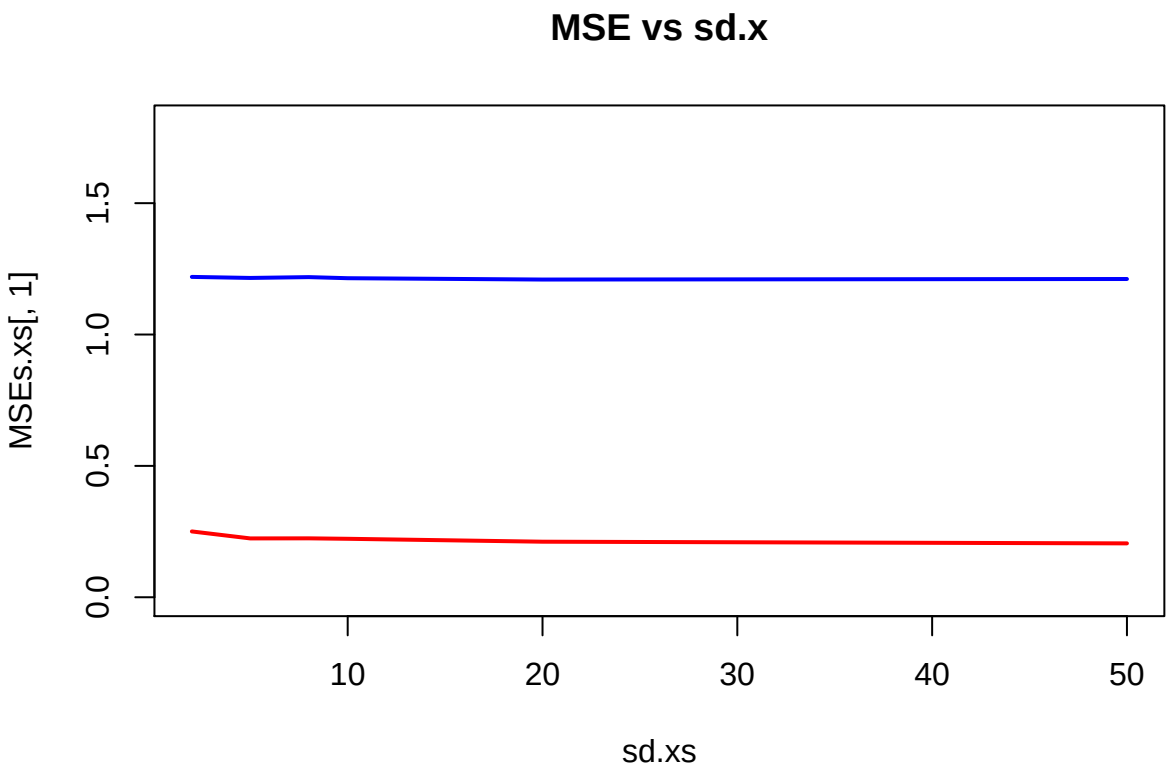
The above plots show a trend in decreasing variance as K grows. However, this gain is likely to be repaid in bias, since we are losing model flexibility as k grows larger.

What if we keep k fixed but change the variance in x?

```
sd.xs <- c(2,5,8,10,20,50)
par(mfrow=c(2,3))
MSEs.xs <- matrix(nrow = 6, ncol=2)
for(i in 1:length(sd.xs)){
  sd.x <- sd.xs[i]
  MSEs <- simulate.knn(100, 8, sd.x = sd.x)
  plot(1:100, MSEs[,1], type="l", main=paste("MSE for sd.x=",sd.x), col="blue", ylim = c(0,2.5))
  abline(h=mean(MSEs[,1]), lty="dashed")
  abline(h=mean(MSEs[,2]), col="red")
  MSEs.xs[i,1] <- mean(MSEs[,1])
  MSEs.xs[i,2] <- mean(MSEs[,2])
}
```

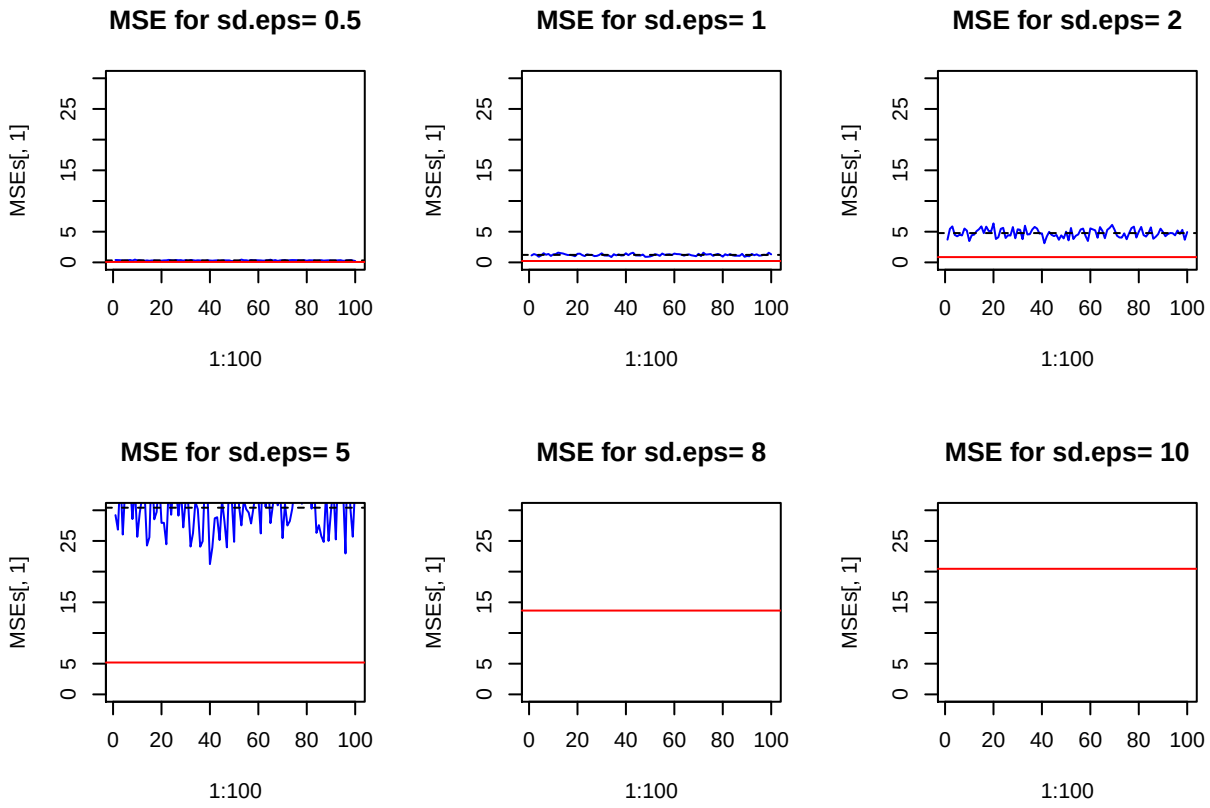



```
par(mfrow=c(1,1))
plot(sd.xs, MSEs.xs[,1], type="l", main = "MSE vs sd.x", col="blue", lwd=2, ylim=c(0,1.8))
points(sd.xs, MSEs.xs[,2], type="l", col="red", lwd=2)
```

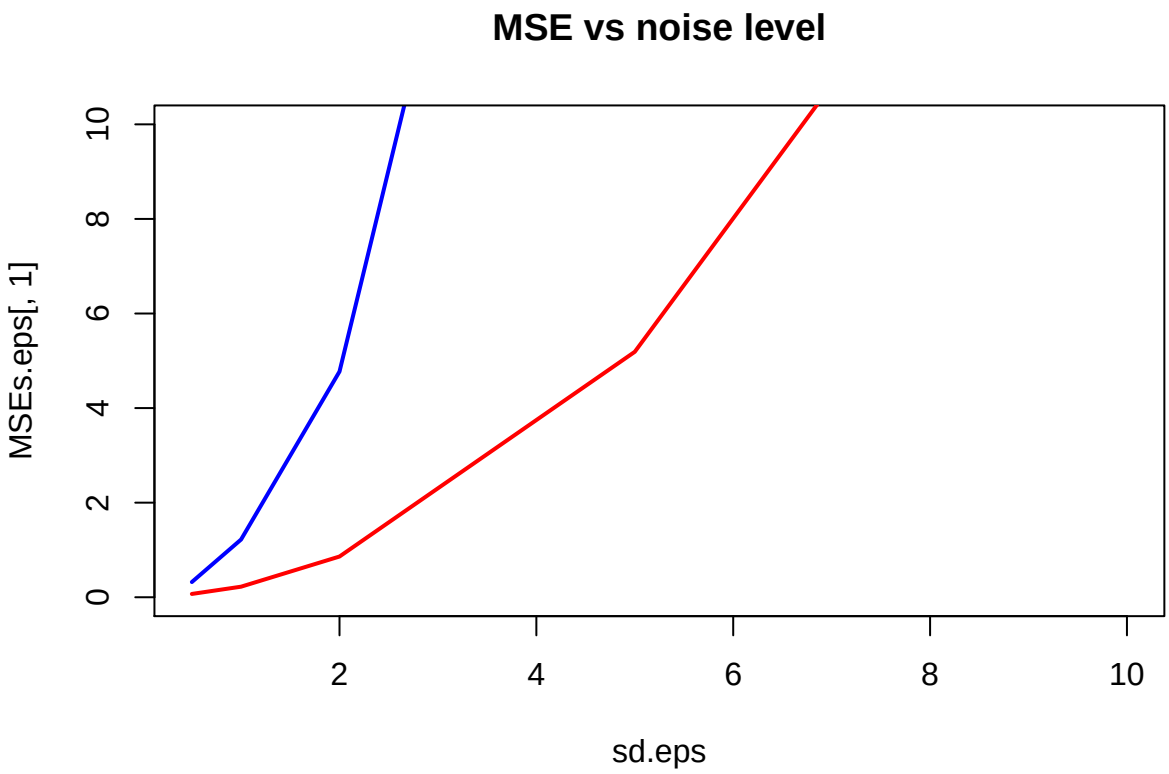


Last but not least, what if we change the noise level?

```
sd.eps <- c(0.5,1,2,5,8,10)
par(mfrow=c(2,3))
MSEs.eps <- matrix(nrow = 6, ncol=2)
for(i in 1:length(sd.eps)){
  sd.ep <- sd.eps[i]
  MSEs <- simulate.knn(100, 8, sd.eps  = sd.ep)
  plot(1:100, MSEs[,1], type="l", main=paste("MSE for sd.eps=",sd.ep), col="blue", ylim = c(0,30))
  abline(h=mean(MSEs[,1]), lty="dashed")
  abline(h=mean(MSEs[,2]), col="red")
  MSEs.eps[i,1] <- mean(MSEs[,1])
  MSEs.eps[i,2] <- mean(MSEs[,2])
}
```




```
par(mfrow=c(1,1))
plot(sd.eps, MSEs.eps[,1], type="l", main = "MSE vs noise level", col="blue", lwd=2, ylim=c(0,10))
points(sd.eps, MSEs.eps[,2], type="l", col="red", lwd=2)
```



Comparing the last plot with the previous section's plots it emerges that the noise level has a much stronger effect than the Xs' sd on the MSE.

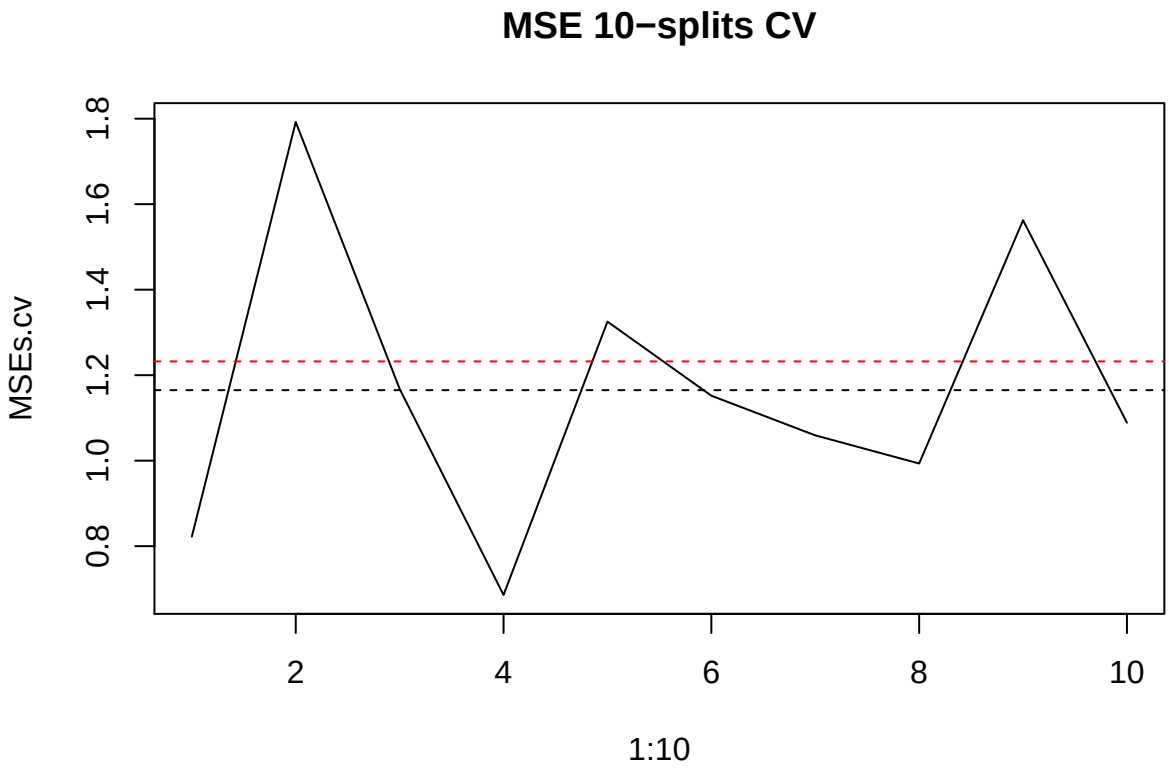
The above approach to compute the MSE is what is called the *validation split approach*. Let's now turn to cross validation, which allows us to gain both in terms of variance and bias of the MSE estimate, without modifying the model.

```
knn.cv <- function(data, nfolds, k){
  n <- nrow(data)
  #shuffling
  s <- sample(1:n, size=n, replace = F)
  folds <- cut(1:n, breaks =nfolds, labels = F)
  MSEs<- matrix(nrow = nfolds, ncol=1)
  for(f in 1:nfolds){
    fold <- s[folds==f]
    train <- data[-fold,]
    test <- data[fold,]
    fit <- kknn(Y~., train = train, test = test, k=k)
    preds <- predict(fit)
    MSE <- sum((test$Y - preds)**2)/length(fold)
    MSEs[f]<-MSE
  }
  return(MSEs)
}
```

```
data <- DataGenerator(n=300, p=3, sd.x = 5, sd.eps =1)
test <- DataGenerator(n=300, p=3, sd.x = 5, sd.eps =1)
MSEs.cv <- knn.cv(data, nfolds = 10, k=8)
```

```
whole.model <- kknn(Y~., train = data, test = test, k=8)
true.MSE <- sum((test$Y - whole.model$fitted.values)**2)/300
```

```
plot(1:10,MSEs.cv, type="l",main="MSE 10-splits CV")
abline(h=mean(MSEs.cv), lty="dashed")
abline(h=mean(true.MSE), lty="dashed", col="red")
```



As we could expect, we are underestimating the true MSE.

```
mean(MSEs.cv)
```

```
## [1] 1.164886
```

```
var(MSEs.cv)
```

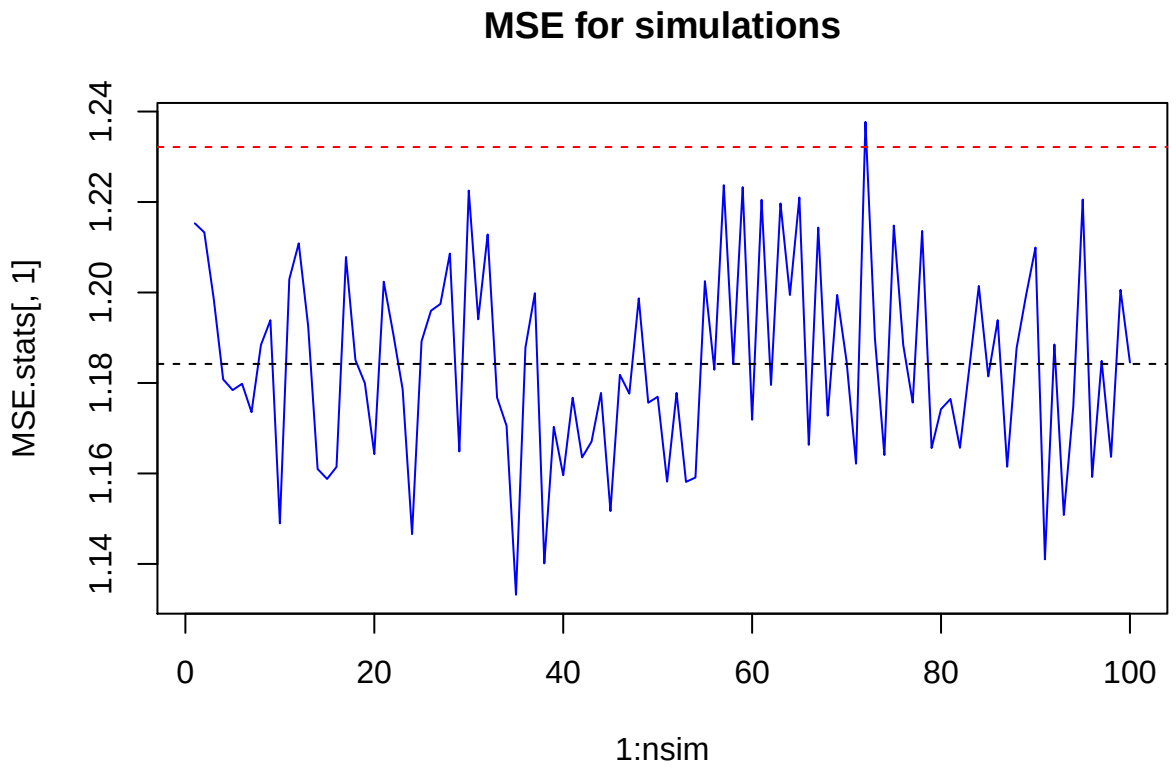
```
##             [,1]
## [1,] 0.1078302
```

Let's now simulate multiple times the cross-validation on the same dataset (only the splits are changing).

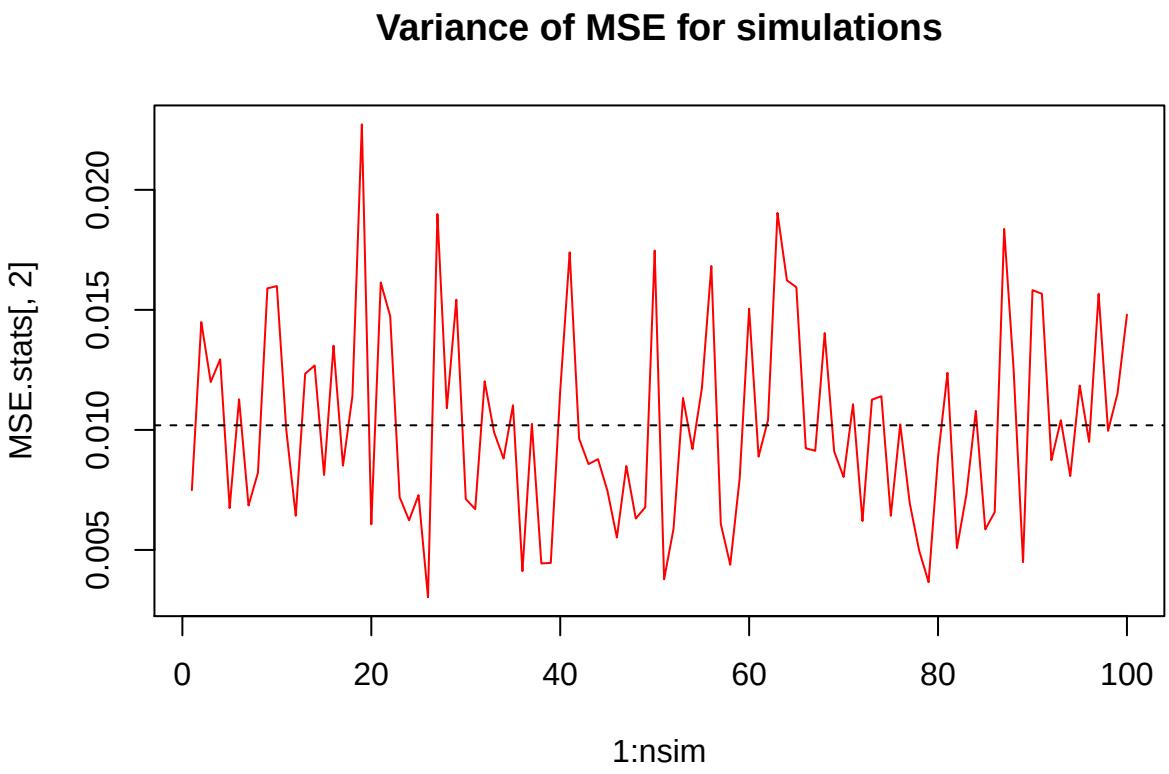
```
nsim <- 100
MSE.stats <- matrix(nrow = nsim, ncol = 2)
for(i in 1:nsim){
  MSEs.cv <- knn.cv(data, nfolds = 10, k=8)
  MSE.stats[i,1] <- mean(MSEs.cv)
  MSE.stats[i,2] <- var(MSEs.cv)/10
}
```

```
plot(1:nsim, MSE.stats[,1], type="l", col="blue", main="MSE for simulations")
abline(h=mean(MSE.stats[,1]), lty="dashed")
```

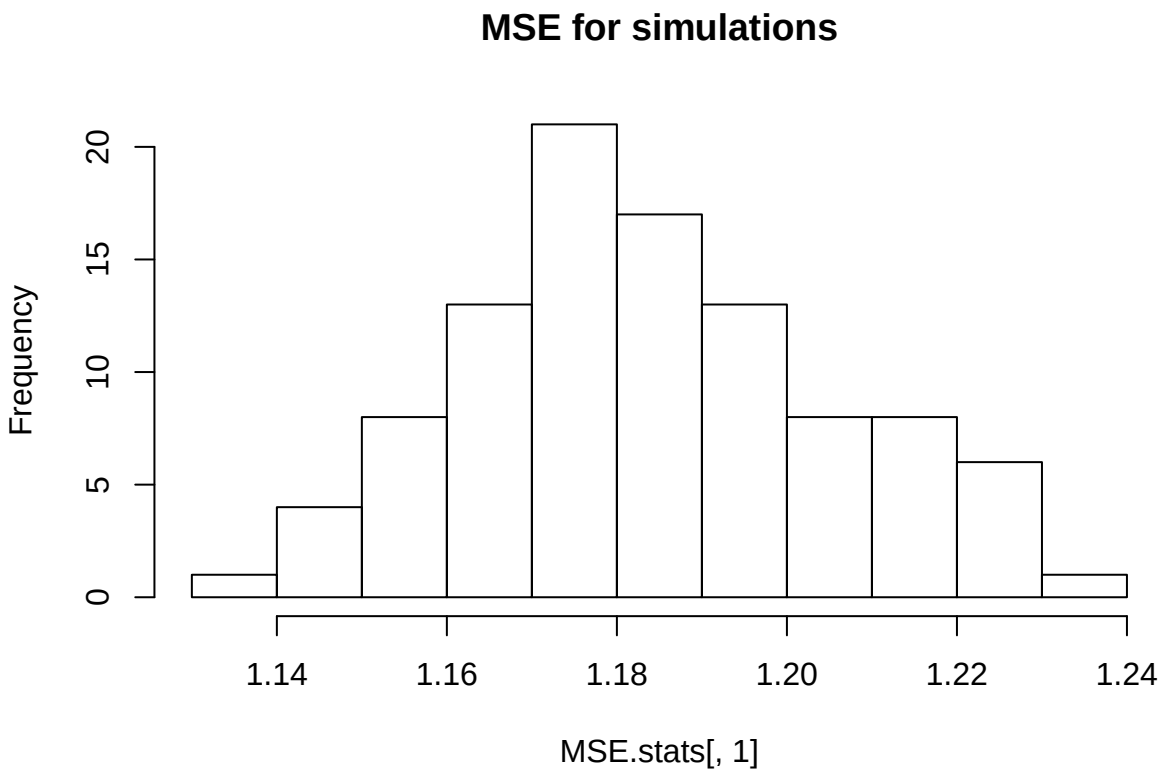
```
abline(h=mean(true.MSE), lty="dashed", col="red")
```



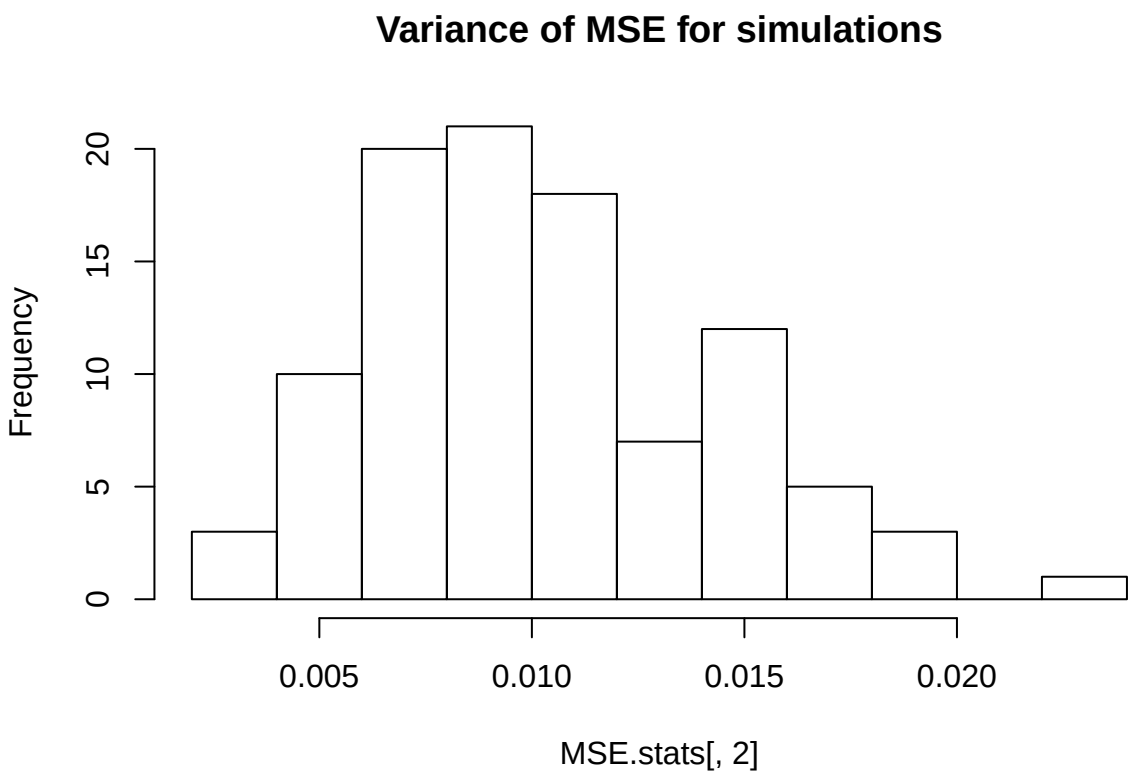

```
plot(1:nsim, MSE.stats[,2], type="l", col="red", main="Variance of MSE for simulations")
abline(h=mean(MSE.stats[,2]), lty="dashed")
```




```
hist(MSE.stats[,1], main="MSE for simulations")
```



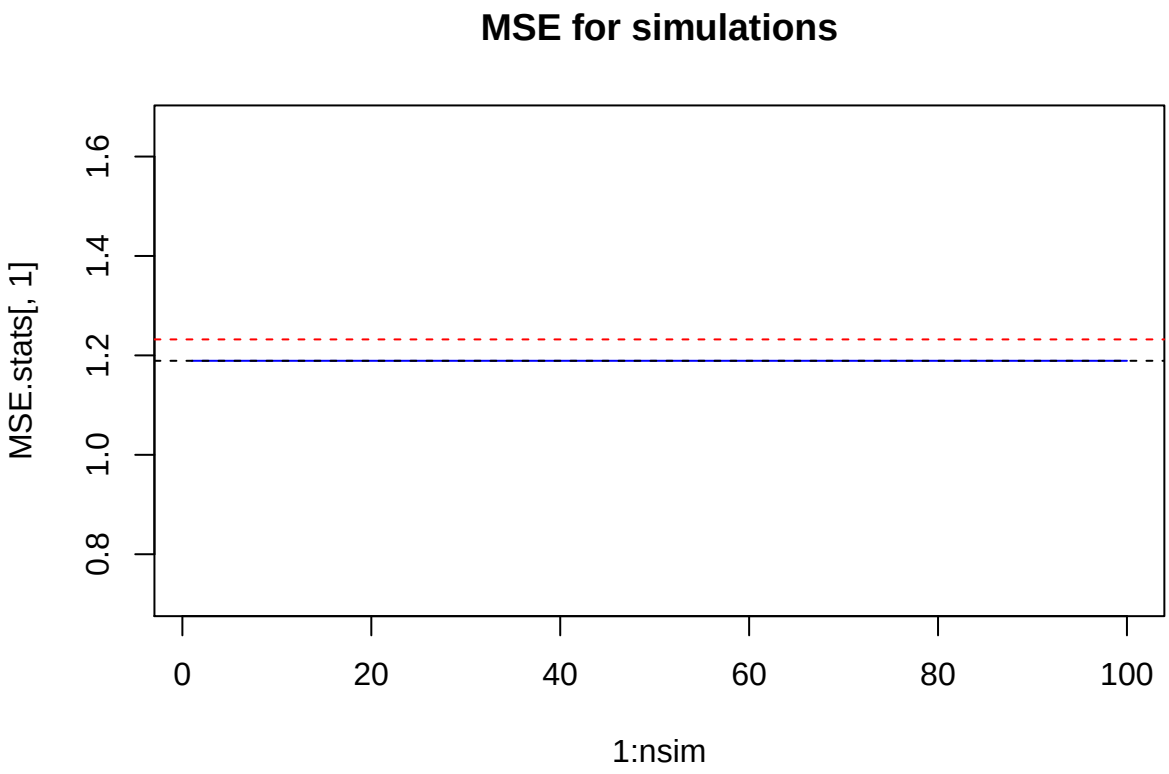

```r
hist(MSE.stats[,2], main="Variance of MSE for simulations")
```



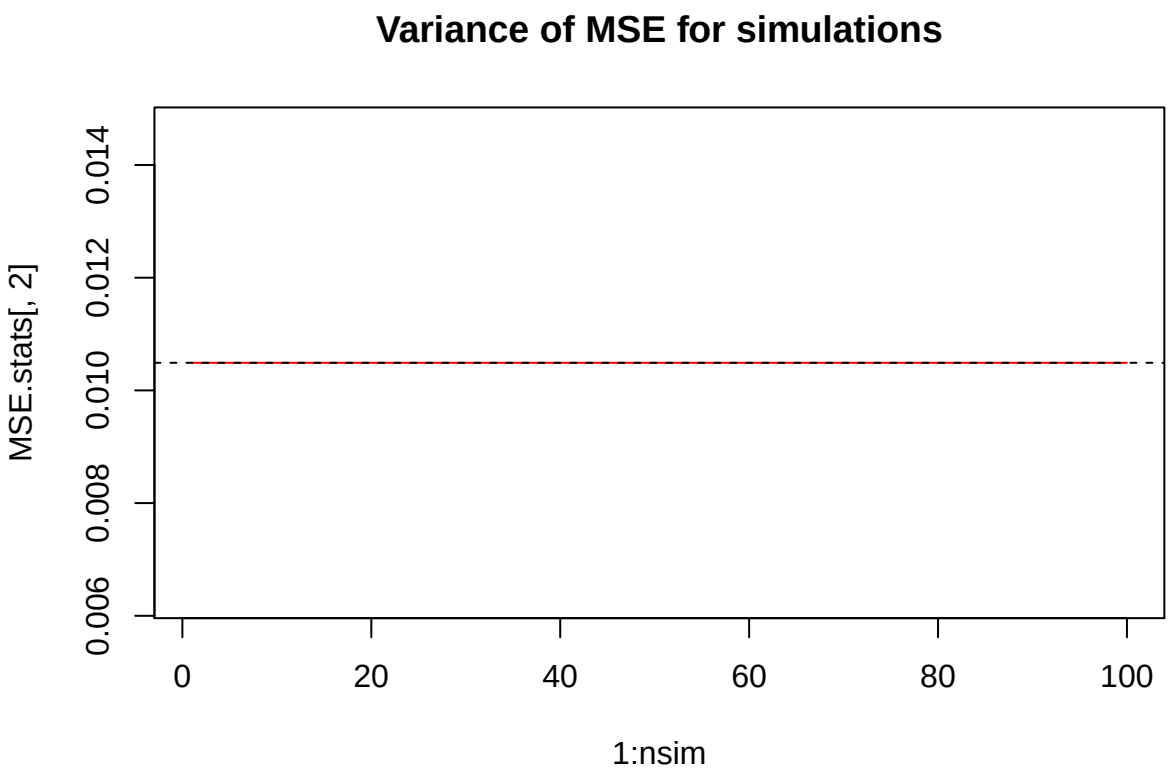
Now let's look at LOOCV to see the effect of the number of splits.

```
nsim <- 100
MSE.stats <- matrix(nrow = nsim, ncol = 2)
for(i in 1:nsim){
  MSEs.cv <- knn.cv(data, nfolds = 300, k=8)
  MSE.stats[i,1] <- mean(MSEs.cv)
  MSE.stats[i,2] <- var(MSEs.cv)/300
}
```

```
plot(1:nsim, MSE.stats[,1], type="l", col="blue", main="MSE for simulations")
abline(h=mean(MSE.stats[,1]), lty="dashed")
abline(h=mean(true.MSE), lty="dashed", col="red")
```




```
plot(1:nsim, MSE.stats[,2], type="l", col="red", main="Variance of MSE for simulations")
abline(h=mean(MSE.stats[,2]), lty="dashed")
```



As expected, we don't get any variance and or bias from the split if we use LOOCV. As you probably have noticed the computational cost of LOOCV is significantly (30x) larger than the k-folds' one.

Model selection + model assessment : how not to do it

We'll know make a huge and common mistake: we'll use the same data for both model selection and model assessment. We'll generate some random dataset with no relationship between the rensponse and the predictors to demonstrate the erroneity of the procedure.

```
set.seed(123)
n <- 50 # sample size
p <- 5000 # nr of predictors
q <- 20 # nr of pre-selected predictors
K <- 10 # nr of folds in cross validation
nr.cv <- 50 # nr of K-fold cross validations that are performed
```

```
# create high-dimensional data
x <- matrix(rnorm(n*p),nrow=n)
y <- c(rep(0,n/2),rep(1,n/2))
# note that x contains no information about y!
```

We'll now select the predictors with highest correlation with the response.

```
select.x <- function(x,y,q){
    # some simple checks on input values:
    stopifnot(is.matrix(x), nrow(x)==length(y), q>=1, q<=ncol(x))
    # compute cor(x[,i],y) for i=1,..,p:
    cor.vec <- apply(x,2,cor,y=y)
```

```
    # determine indices of variables, so that the absolute value of
    # their correlation with y is sorted in decreasing order:
    ind <- order(abs(cor.vec), decreasing=TRUE)
    # return indices of q variables with largest absolute correlation
    return(ind[1:q])
}
```

```
x.new <- x[,select.x(x,y,q)]
```

We'll now use cross validation to assess our selected model. Note: we've already performed model selection on the data on which we're going to perform cross validation. Also note: being the y and the x totally unrelated an honest error should be around 0.5.

```
library(class)
```

```
## Warning: package 'class' was built under R version 3.6.3
```

```
##
## Attaching package: 'class'
```

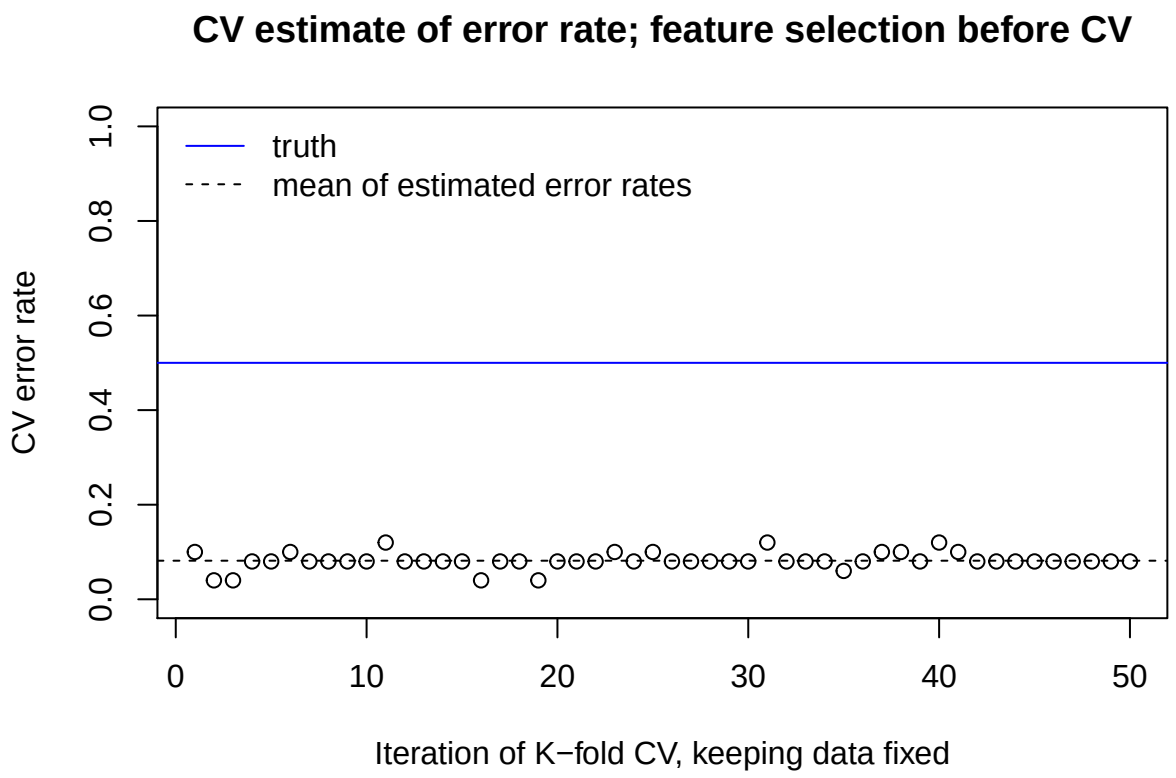
```
## The following object is masked _by_ '.GlobalEnv':
##
##     knn.cv
```

```
cv.knn1 <- function(x,y,K,q=10){
    # quick checks of input:
    stopifnot(is.matrix(x), nrow(x)==length(y), 1<=K, K<=nrow(x),q>=1, q<=ncol(x))

    # randomly shuffle the rows:
    n <- length(y)
    ind.x <- sample(c(1:n), replace=FALSE)
    x <- x[ind.x,]
    y <- y[ind.x]

    # create K (roughly) equally sized folds:
    folds <- cut(seq(1,n),breaks=K,labels=FALSE)

    # perform K fold cross validation:
    error <- integer(K)
    for(i in 1:K){
      # Segment data by fold using the which() function
      ind.test <- which(folds==i)
      x.test <- x[ind.test,]
      y.test <- y[ind.test]
      x.train <- x[-ind.test,]
      y.train <- y[-ind.test]
      y.pred <- knn1(x.train, x.test, y.train)
      error[i] <- sum(y.pred != y.test)
    }
    return(sum(error/n))
}
```

```
# assess performance of 1 NN classifier via K-fold cross validation.
cv.estimation <- replicate(nr.cv, cv.knn1(x.new,y,K=10))
```

```
plot(cv.estimation, ylim=c(0,1), ylab="CV error rate",
     xlab="Iteration of K-fold CV, keeping data fixed",
```

```
     main="CV estimate of error rate; feature selection before CV")
abline(h=mean(cv.estimation),lty=2)
abline(h=0.5,col="blue")
legend("topleft",c("truth","mean of estimated error rates"),
       col=c("blue","black"),lty=c(1,2), bty="n")
```



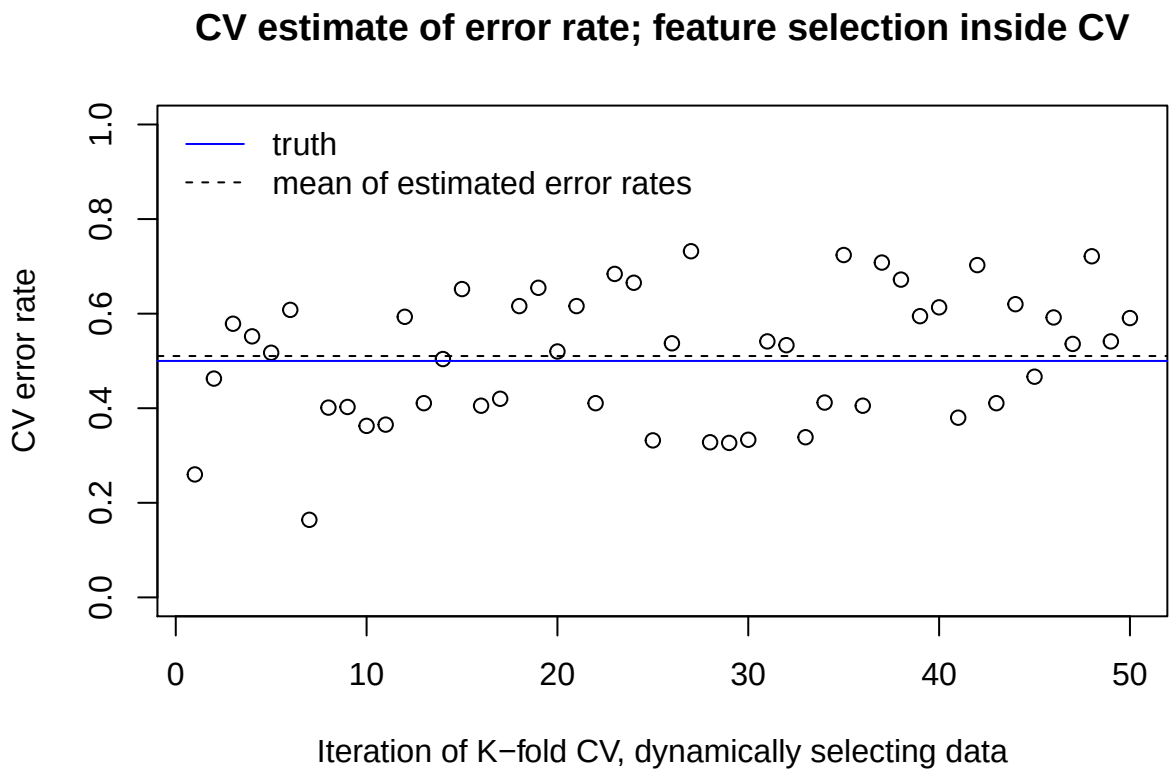
From the plot it's clear we're severly under-estimating our error-rate! To get a realistic estimate we should perform the selection on a training dataset and test on unseen data. Let's do it.

```
selection.assessment <- function(x,y,K,test_split,q=10){
    # quick checks of input:
    stopifnot(is.matrix(x), nrow(x)==length(y), 1<=K, K<=nrow(x),q>=1, q<=ncol(x))

    # randomly shuffle the rows:
    n <- length(y)
    ind.x <- sample(c(1:n), replace=FALSE)
    x <- x[ind.x,]
    y <- y[ind.x]

    # we'll create a test split and a train split
    n.test <- round(n*test_split)
    ind.test <- sample(1:n, size =n.test, replace=F)

    # we'll use the first fold for testing and the rest for training
    x.test <- x[ind.test,]
    y.test <- y[ind.test]
    x.train <- x[-ind.test,]
```

```
    y.train <- y[-ind.test]

    # model selection here
    selected.features<-select.x(x.train,y.train,q)
    x.new.train <- x.train[,selected.features]
    x.new.test <- x.test[,selected.features]

    # and finally cross validation on the test fold to assess the model performance
    cv.estimation <- replicate(nr.cv, cv.knn1(x.new.test,y.test,K=K))
    error <- mean(cv.estimation)

    return(error)
}
```

```
cv.estimation.right <- replicate(nr.cv, selection.assessment(x,y,K=8,test_split = 0.3))
```

```
plot(cv.estimation.right, ylim=c(0,1), ylab="CV error rate",
     xlab="Iteration of K-fold CV, dynamically selecting data",
     main="CV estimate of error rate; feature selection inside CV")
abline(h=mean(cv.estimation.right),lty=2)
abline(h=0.5,col="blue")
legend("topleft",c("truth","mean of estimated error rates"),
       col=c("blue","black"),lty=c(1,2), bty="n")
```



Now the error estimate is close to the truth.