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        X.6
## 1 -0.3357457 -12.3598520  0.33820420 -5.1969544 -5.432351 -0.6556808 -4.8897815
## 2  0.2183649  -8.7846196  5.46791875 -0.6040204 -7.258462 -3.2024614  3.4026288
## 3 -1.8622345   2.3492732  6.11887876  5.7701738  5.451211 -5.1131953  0.2636945
## 4 -1.4096098  -0.2088155  4.65372818  1.2882619 -1.582932 -6.7816822  1.1159976
## 5 -1.0306837   3.9377403 -6.46744714  3.6501032  5.136073  9.1971863  1.0767955
## 6 -0.1091457   1.2272880 -0.05954332 -2.2189770 -4.401342  2.8335466 -3.2794982
##         X.7       X.8       X.9        X.10
## 1 -4.476856 -4.413228 -6.842902 -4.22384289
## 2  1.692396 -7.078793  2.960945  0.02587641
## 3 -2.005542  4.467687 -5.027812 -7.12393019
## 4 -3.567620 -2.854977  8.050231 -2.08049431
## 5 -5.696733 -2.489927  5.400199  1.75358639
## 6 -3.992537  9.540822  3.366155  2.65216676
```

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

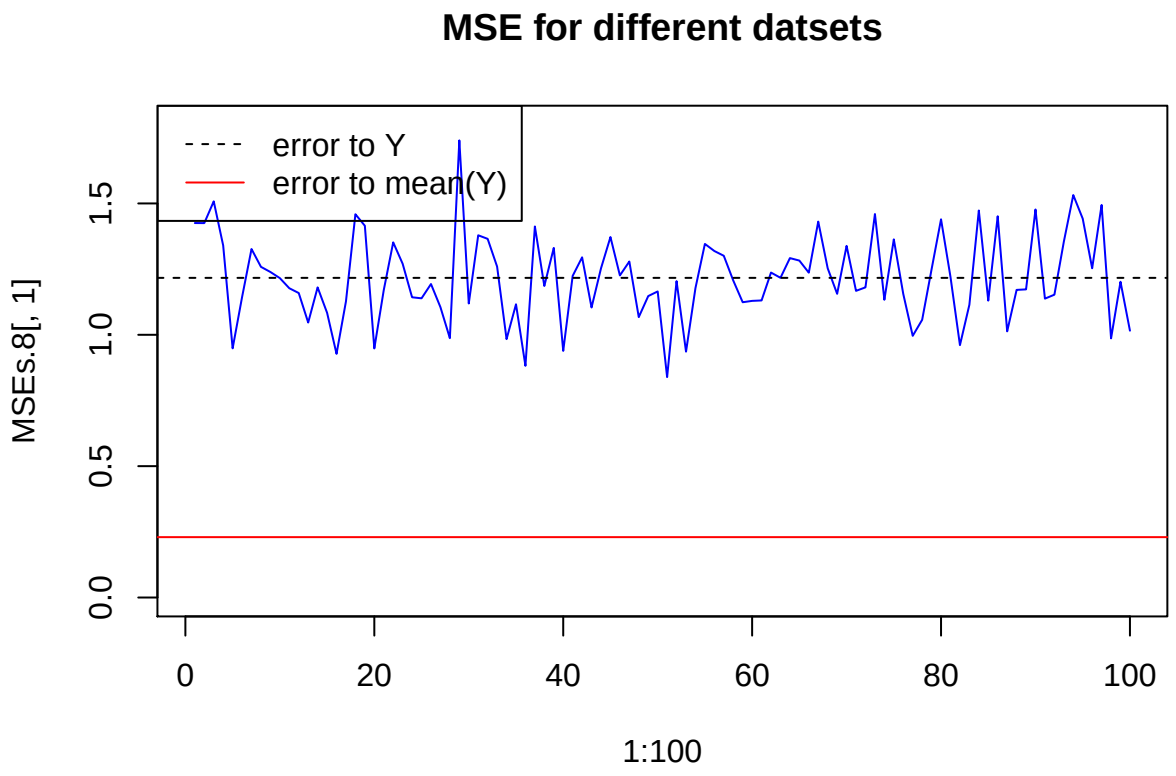
```
## [1] 1.314675
```

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)
}
```

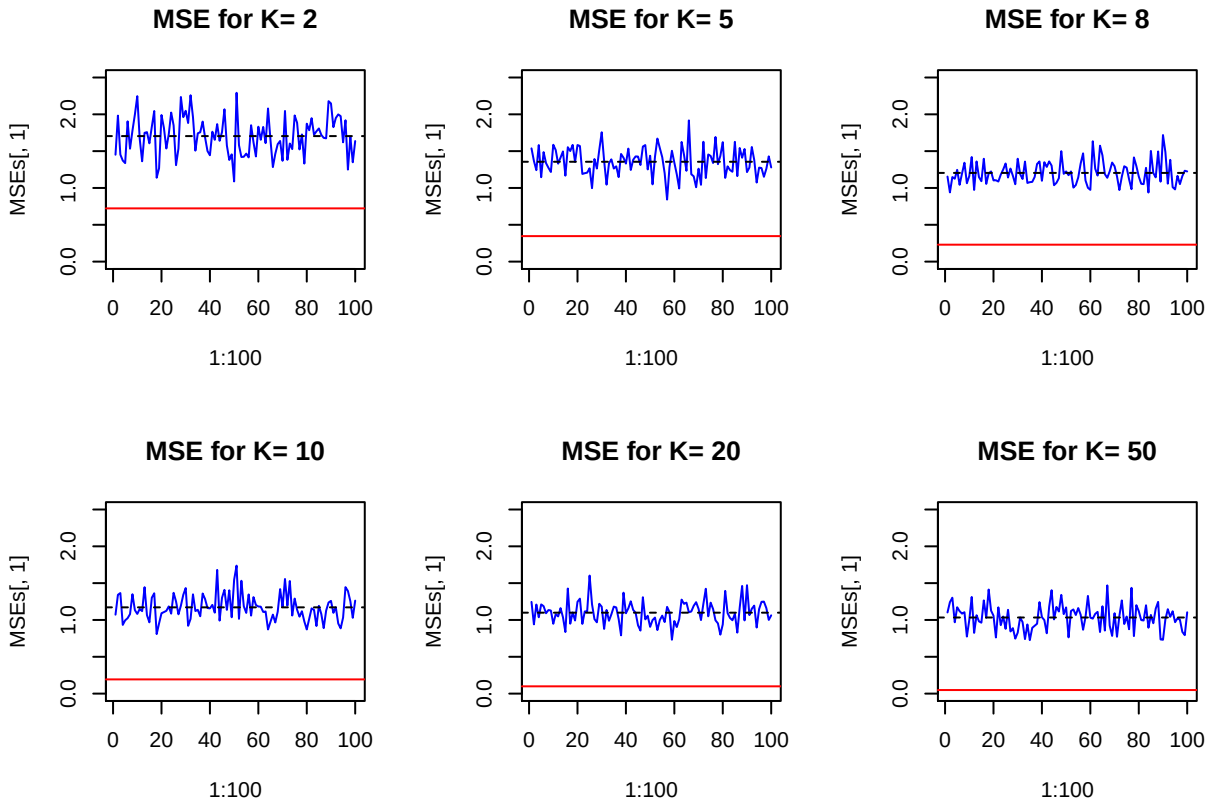
```
MSEs.8 <- simulate.knn(100, 8)
plot(1:100, MSEs.8[,1], type="l", main="MSE for different datsets", col="blue", ylim = c(0,1.8))
abline(h=mean(MSEs.8[,1]), lty="dashed")
abline(h=mean(MSEs.8[,2]), col="red")
legend("topleft", legend = c("error to Y","error to mean(Y)"), col = c("black","red"),lty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



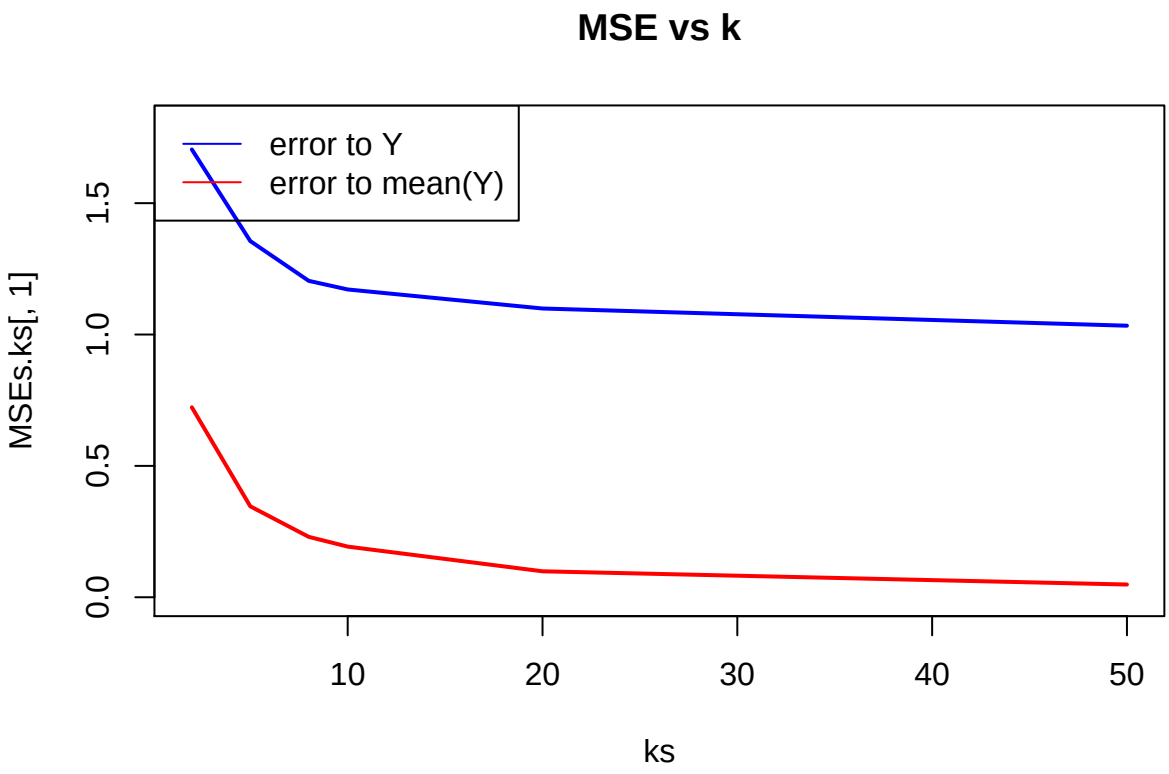
Now let's investigate how the MSE would change for different k.

```
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])
}
```



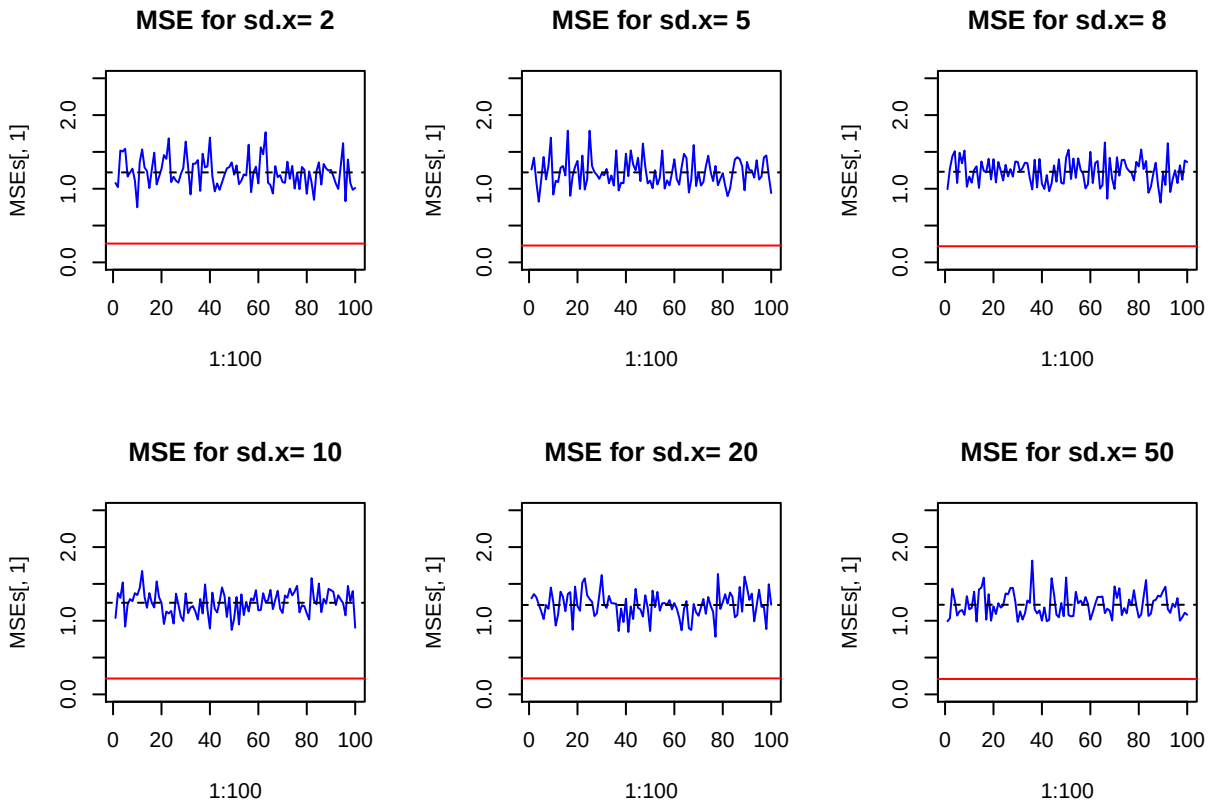

```
par(mfrow=c(1,1))
plot(ks, MSEs.ks[,1], type="l", main = "MSE vs k", col="blue", lwd=2, ylim=c(0,1.8))
points(ks, MSEs.ks[,2], type="l", col="red", lwd=2)
legend("topleft", legend = c("error to Y","error to mean(Y)"), col = c("blue","red"),lty =c("solid","so
```



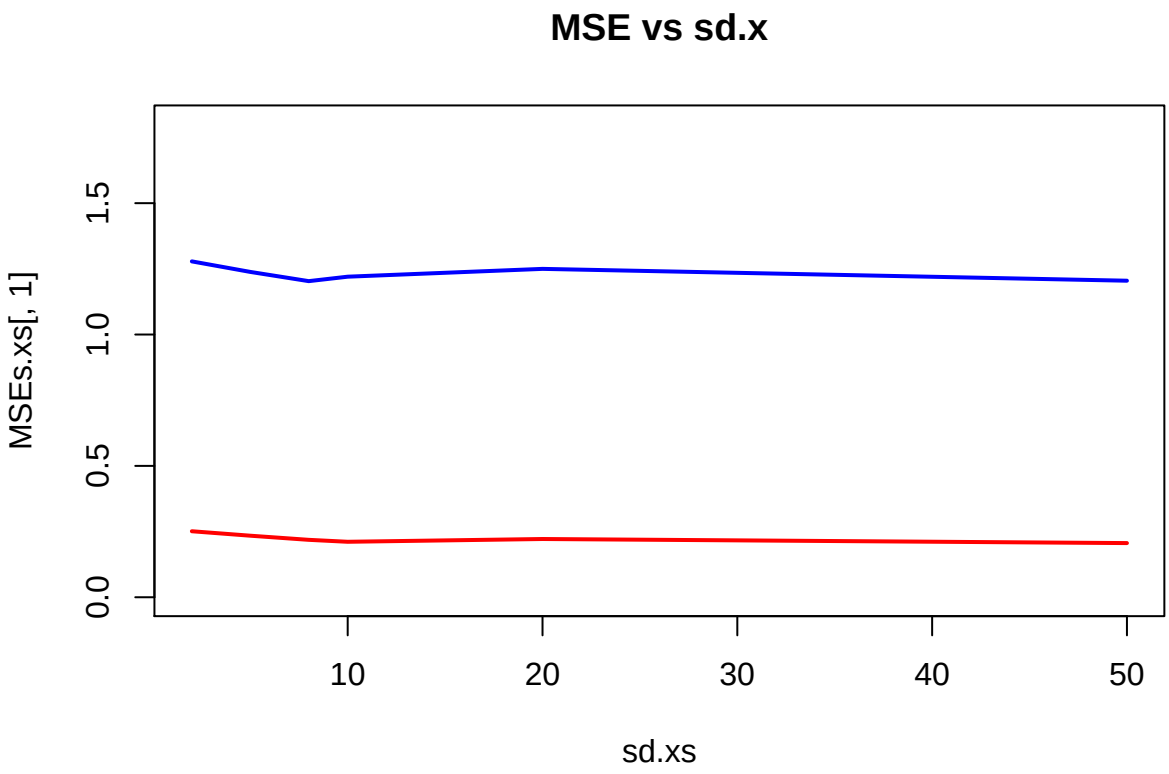
The above plots show a trend in decreasing variance as K grows. However, this gain is likely to be repaid in bias, since we are losing model flexibility as k grows larger.

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])
}
```

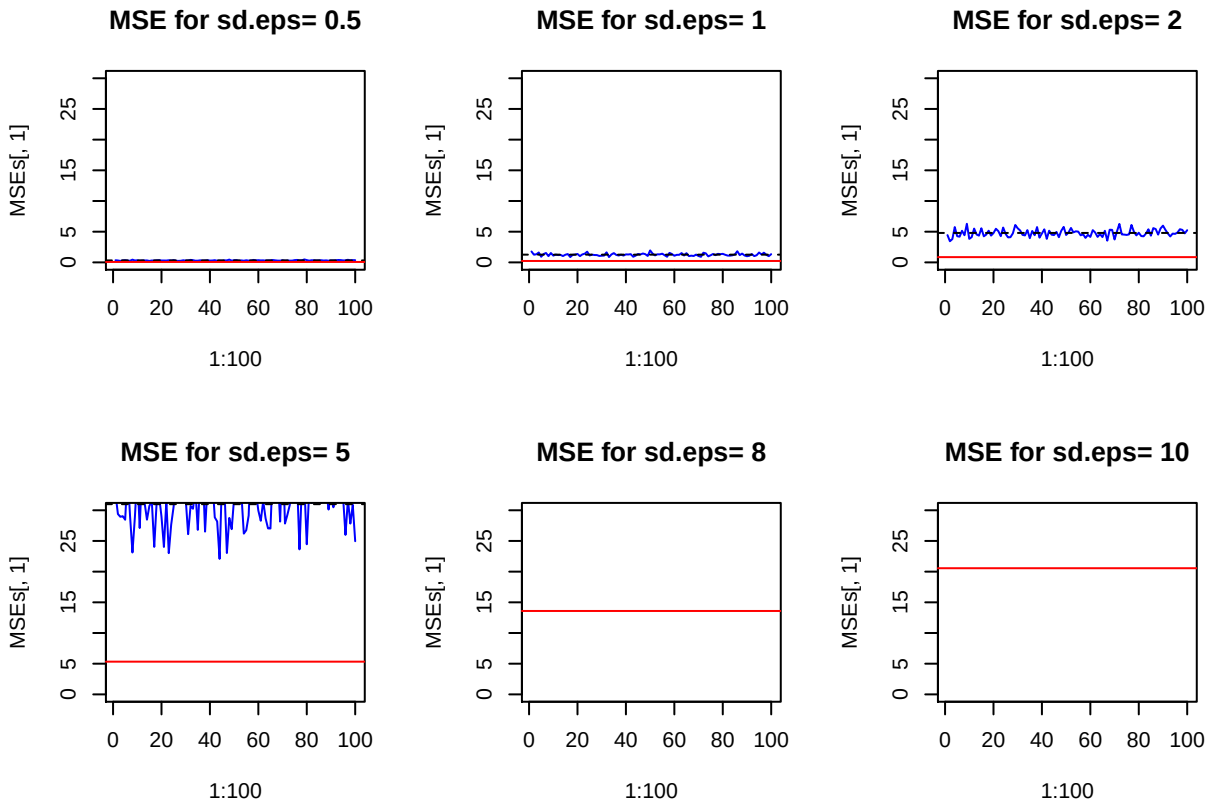



```
par(mfrow=c(1,1))
plot(sd.xs, MSEs.xs[,1], type="l", main = "MSE vs sd.x", col="blue", lwd=2, ylim=c(0,1.8))
points(sd.xs, MSEs.xs[,2], type="l", col="red", lwd=2)
```

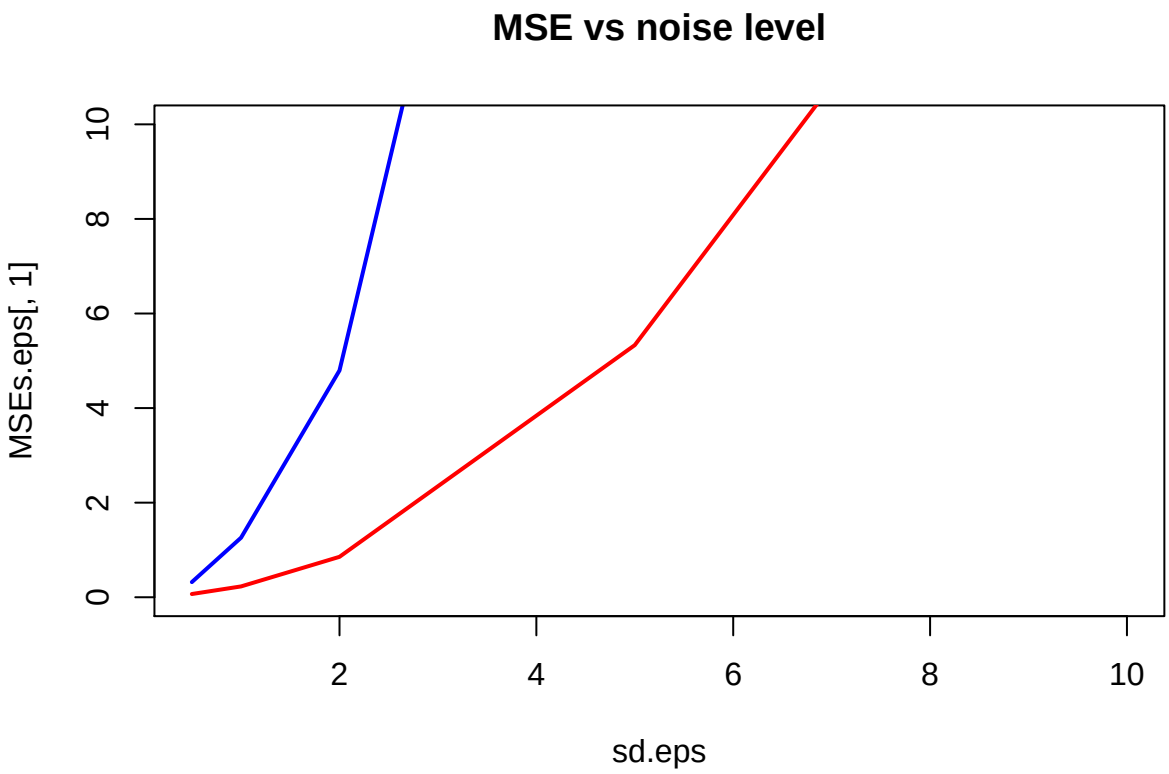


Last but not least, what if we change the noise level?

```
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.

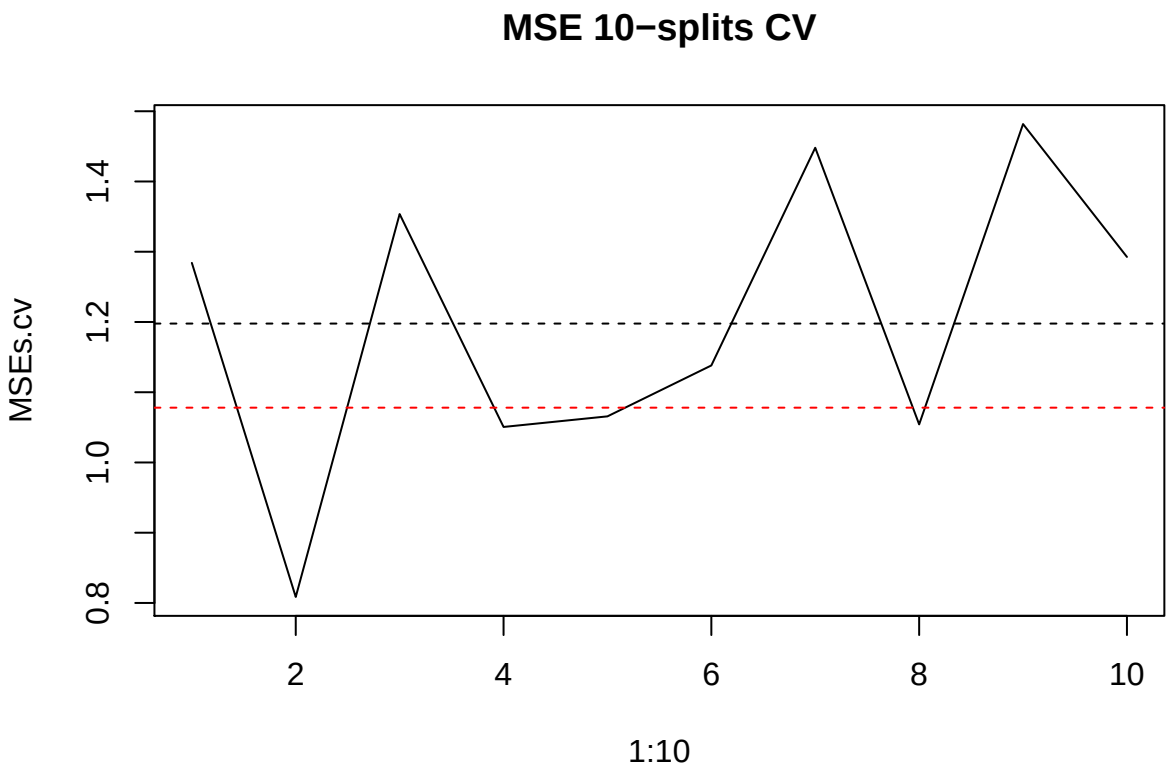
The above approach to compute the MSE is what is called the *validation split approach*. Let's now turn to cross validation, which allows us to gain both in terms of variance and bias of the MSE estimate, without modifying the model.

```
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] 1.197725
```

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

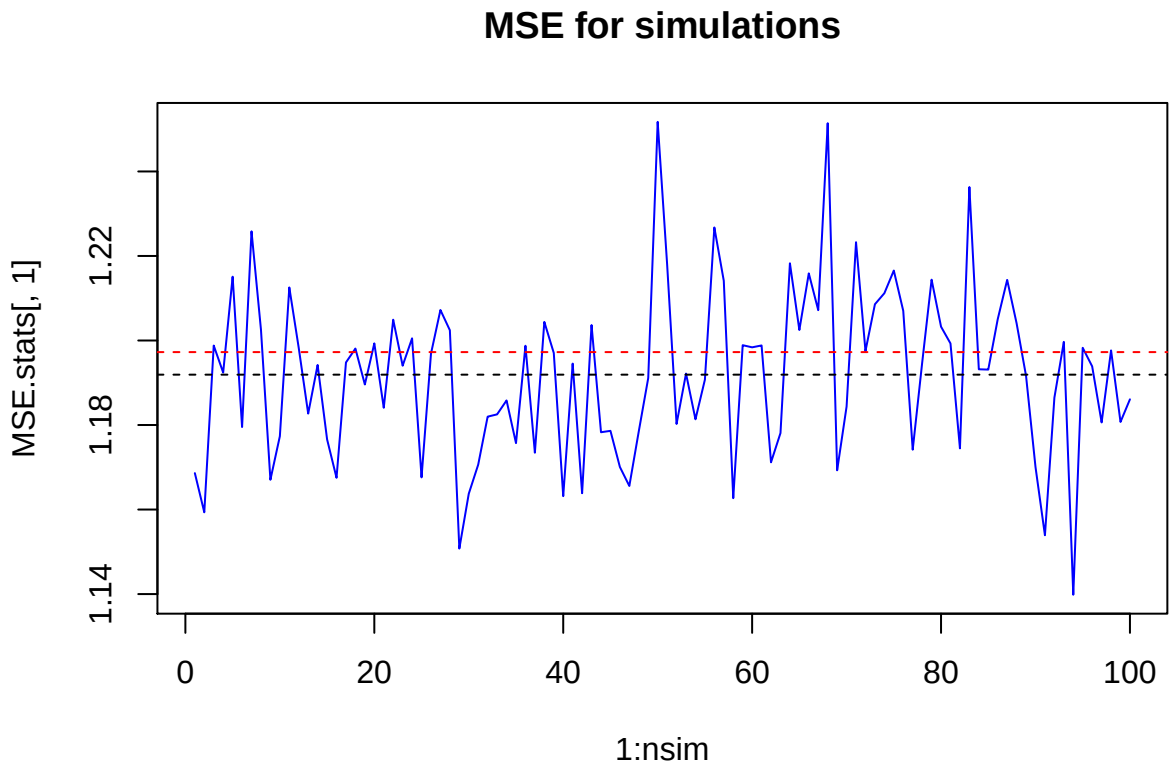
```
##              [,1]
## [1,] 0.04431853
```

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

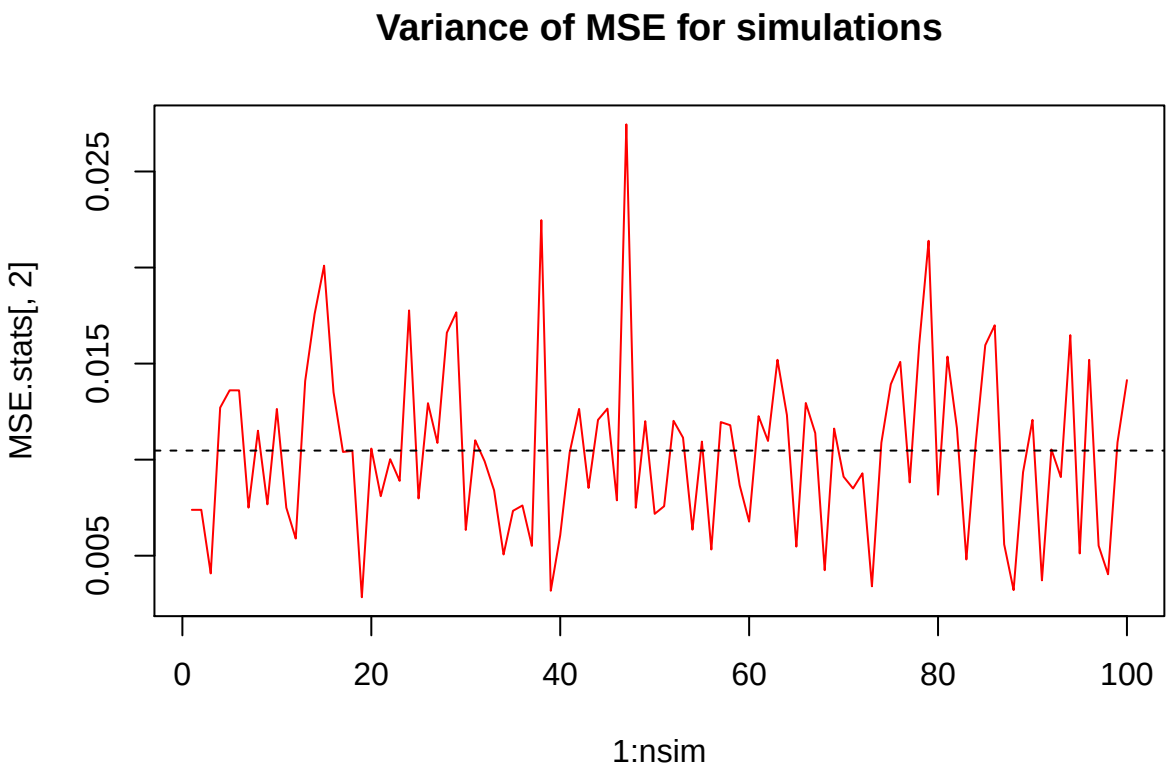
```
nsim <- 100
MSE.stats <- matrix(nrow = nsim, ncol = 2)
for(i in 1:nsim){
  MSEs.cv <- knn.cv(data, nfolds = 10, k=8)
  MSE.stats[i,1] <- mean(MSEs.cv)
  MSE.stats[i,2] <- var(MSEs.cv)/10
}
```

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

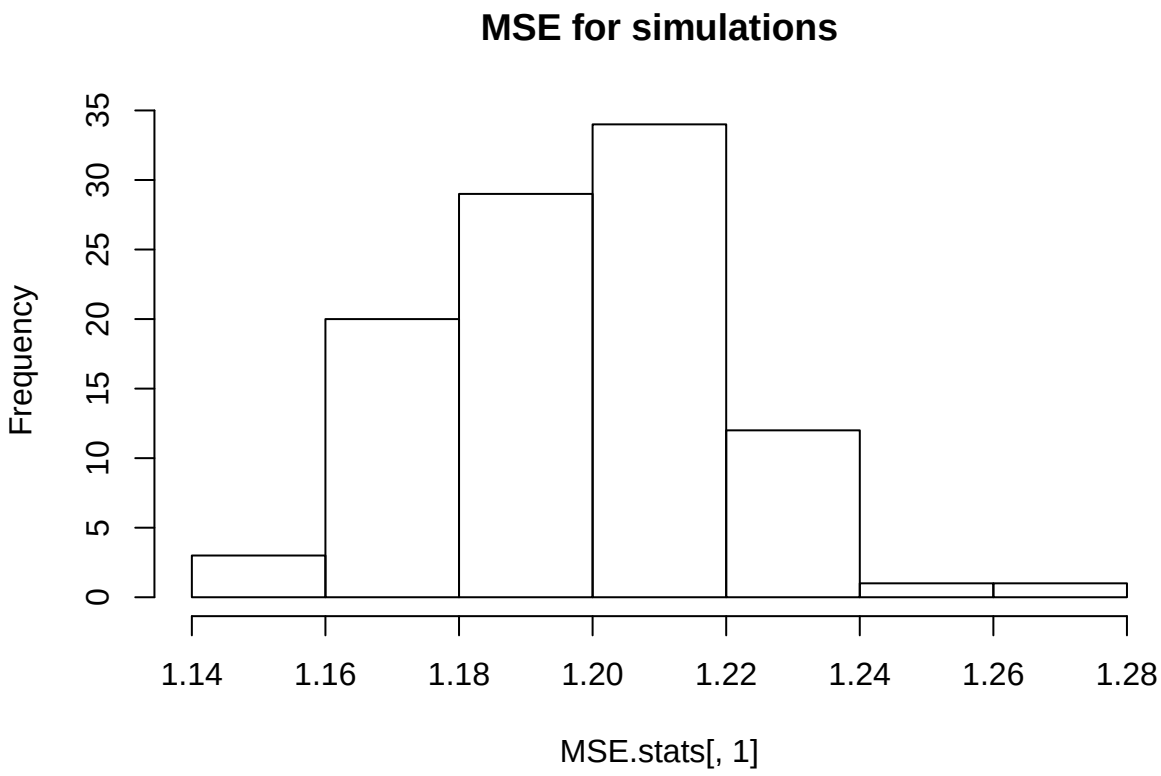
```
abline(h=mean(true.MSE), lty="dashed", col="red")
```



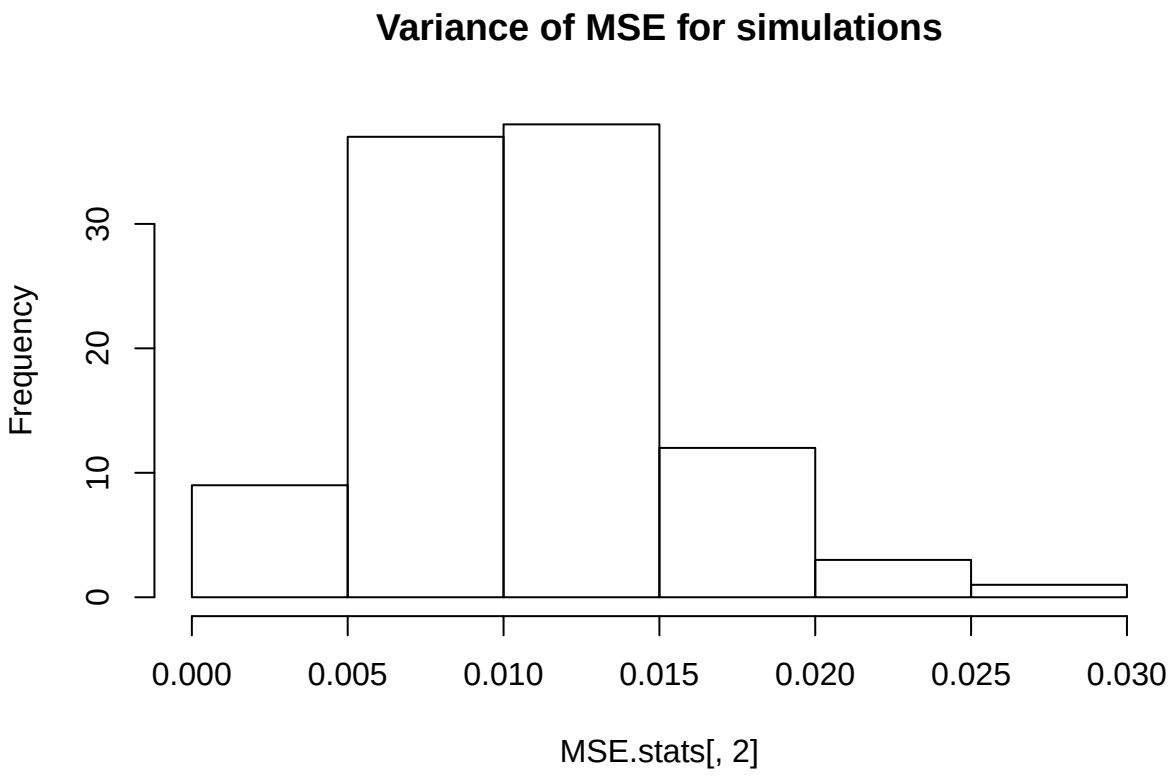

```
plot(1:nsim, MSE.stats[,2], type="l", col="red", main="Variance of MSE for simulations")
abline(h=mean(MSE.stats[,2]), lty="dashed")
```




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



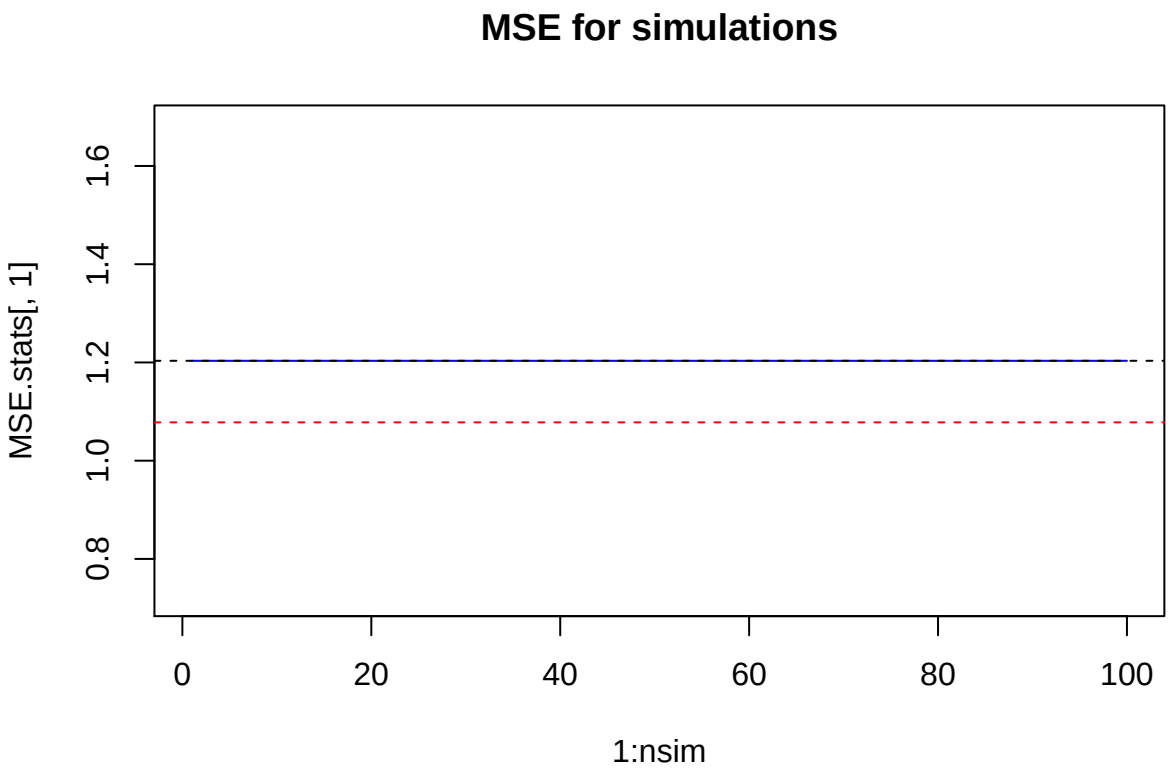

```
hist(MSE.stats[,2], main="Variance of MSE for simulations")
```



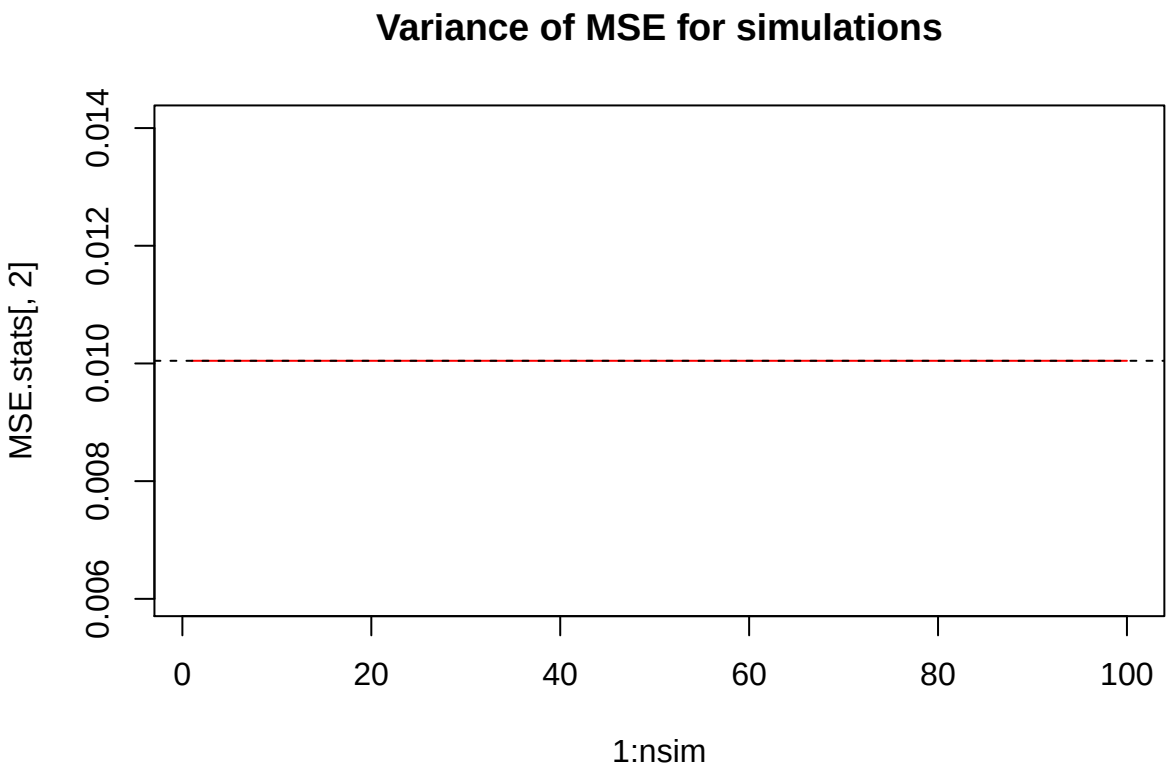
Now let's look at LOOCV to see the effect of the number of splits.

```
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'
```

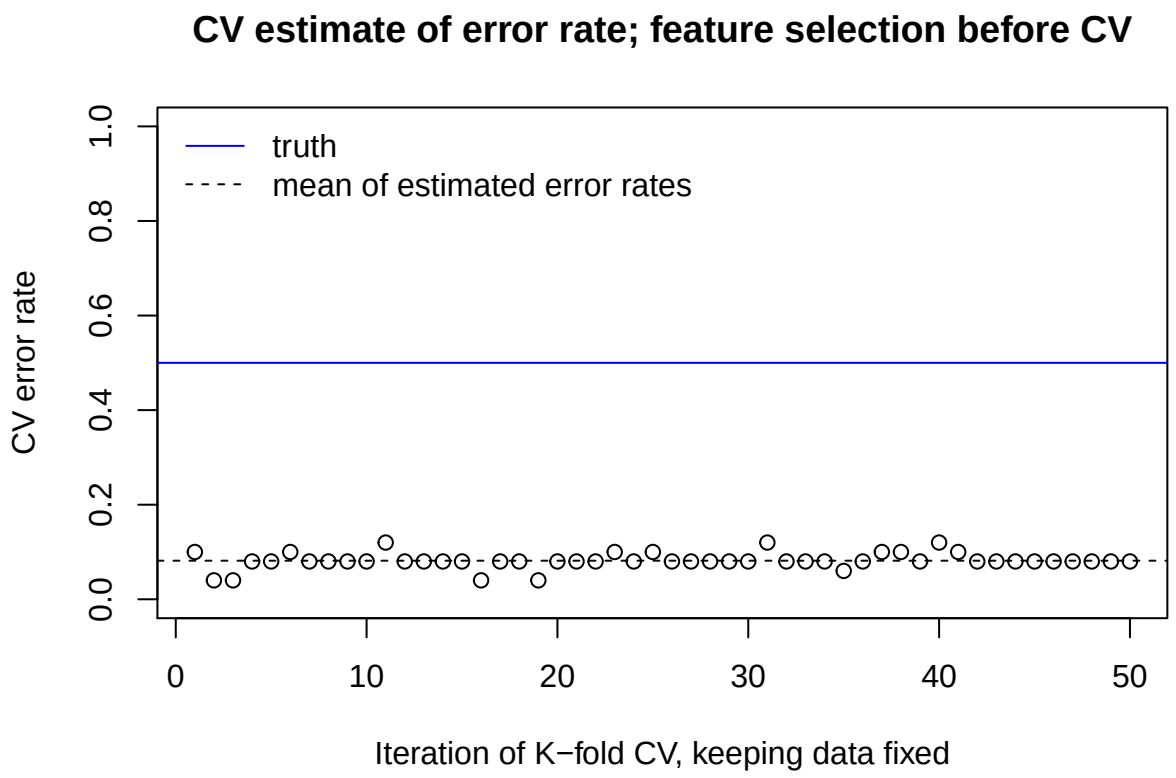
```
## The following object is masked _by_ '.GlobalEnv':
##
##     knn.cv
```

```
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")
```



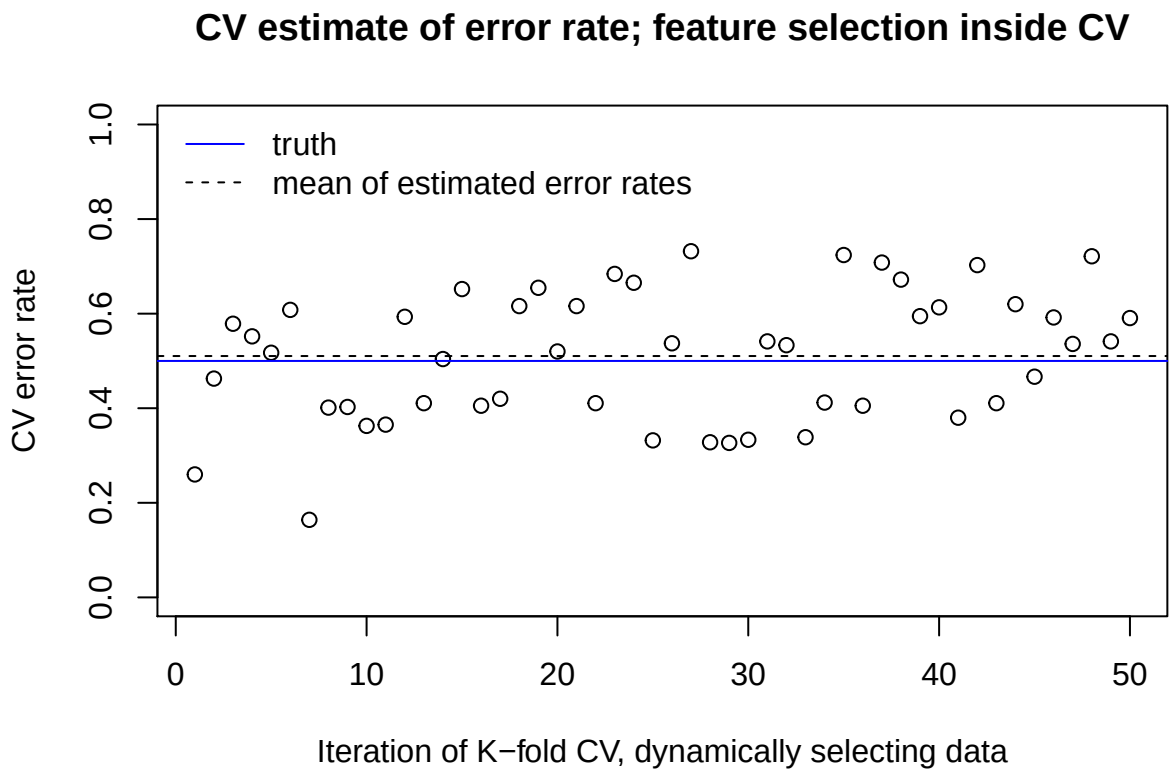
From the plot it's clear we're severly under-estimating our error-rate! To get a realistic estimate we should perform the selection on a training dataset and test on unseen data. Let's do it.

```
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.

We'll now use cross-validation to assess a classification model. The data we'll work on is the Weekly data. Weekly percentage returns for the S&P 500 stock index between 1990 and 2010.

```
require("ISLR")
```

```
## Loading required package: ISLR
```

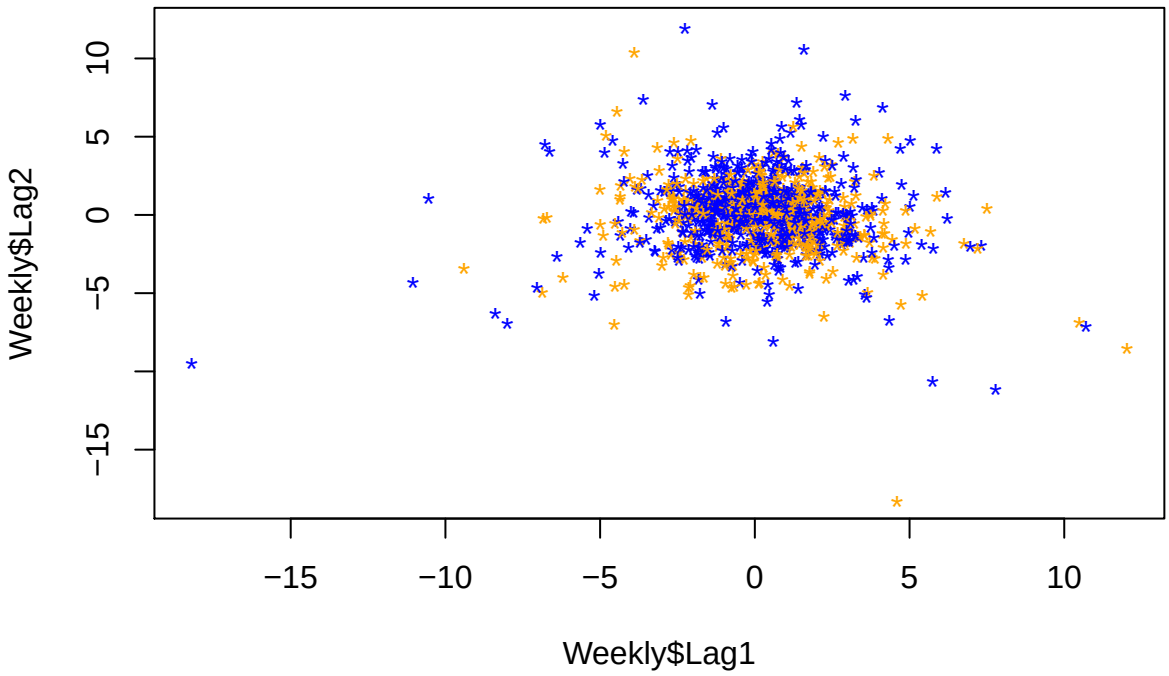
```
## Warning: package 'ISLR' was built under R version 3.6.3
```

```
head(Weekly)
```

```
##   Year   Lag1   Lag2   Lag3   Lag4   Lag5    Volume  Today Direction
## 1 1990  0.816  1.572 -3.936 -0.229 -3.484 0.1549760 -0.270      Down
## 2 1990 -0.270  0.816  1.572 -3.936 -0.229 0.1485740 -2.576      Down
## 3 1990 -2.576 -0.270  0.816  1.572 -3.936 0.1598375  3.514        Up
## 4 1990  3.514 -2.576 -0.270  0.816  1.572 0.1616300  0.712        Up
## 5 1990  0.712  3.514 -2.576 -0.270  0.816 0.1537280  1.178        Up
## 6 1990  1.178  0.712  3.514 -2.576 -0.270 0.1544440 -1.372      Down
```

We'll fit a logistic regressionmodel that predicts Direction using Lag1 and Lag2.

```
plot(Weekly$Lag1, Weekly$Lag2, col=ifelse(Weekly$Direction=="Down","orange","blue"), pch="*")
```




```
logistic <- glm(Direction~Lag1+Lag2, family ="binomial", data =Weekly)
summary(logistic)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2, family = "binomial", data = Weekly)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
```

```
## -1.623  -1.261   1.001   1.083   1.506
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.22122    0.06147   3.599 0.000319 ***
## Lag1        -0.03872    0.02622  -1.477 0.139672
## Lag2         0.06025    0.02655   2.270 0.023232 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1496.2  on 1088  degrees of freedom
## Residual deviance: 1488.2  on 1086  degrees of freedom
## AIC: 1494.2
##
## Number of Fisher Scoring iterations: 4
```

Now let's fit again the logistic model using all observations but the first.

```
logistic.1 <- glm(Direction~Lag1+Lag2, family ="binomial", data=Weekly[-1,])
logistic.1
```

```
##
## Call:  glm(formula = Direction ~ Lag1 + Lag2, family = "binomial", data = Weekly[-1,
##     ])
##
## Coefficients:
## (Intercept)         Lag1         Lag2
##     0.22324     -0.03843      0.06085
##
## Degrees of Freedom: 1087 Total (i.e. Null);  1085 Residual
## Null Deviance:       1495
## Residual Deviance: 1487  AIC: 1493
```

The coefficients are almost the same. Hence the first observation doesn't have high leverage on the dataset. Let's look at our error on the first observation.

```
# let's look at the encoding of the output in the model:
levels(Weekly$Direction)
```

```
## [1] "Down" "Up"
```

```
# being Down the first level it will be encoded as 0
# Hence the prediction "Up" corresponds to a probability higher than 0.5
ifelse(predict(logistic.1, newdata=Weekly[1,])>0.5,"Up","Down")
```

```
##      1
## "Down"
```

The prediction is correct, and the confidence was:

```
1 - predict(logistic.1, newdata=Weekly[1,])
```

```
##        1
## 0.712466
```

Let's repeat this process n times (one for each observation).

```
n <- dim(Weekly)[1]
errors <- rep(0,n)
for(i in 1:n){
  logistic.i <- glm(Direction~Lag1+Lag2, family ="binomial", data=Weekly[-i,])
  prediction <- ifelse(predict(logistic.i, newdata=Weekly[i,])>0.5,"Up","Down")
  error <- prediction!=Weekly[i,]$Direction
  errors[i]<-error
}
```

```
(mean.error.LOOCV <- mean(errors))
```

```
## [1] 0.5454545
```

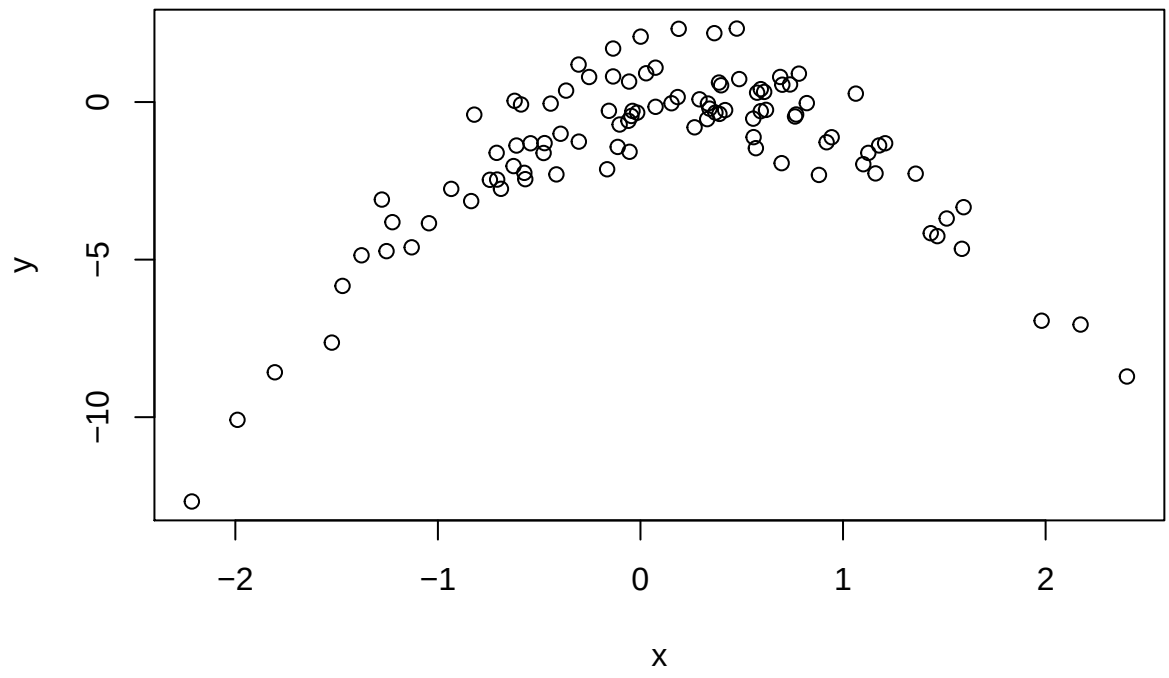
```
(var.error.LOOCV <- var(errors))
```

```
## [1] 0.2481618
```

The results are not surprising since from the data visualisation above it seemed clearly not linearly separable.

We will now perform cross-validation on a simulated data set.

```
set.seed(1)
x <- rnorm(100)
y=x-2* x^2+ rnorm (100)
plot(x,y)
```




```
data <- data.frame(X=x, Y=y)
head(data)
```

```
##            X          Y
## 1 -0.6264538 -2.0317092
## 2  0.1836433  0.1583095
## 3 -0.8356286 -3.1431006
## 4  1.5952808 -3.3365321
## 5  0.3295078 -0.5422276
## 6 -0.8204684 -0.3995179
```

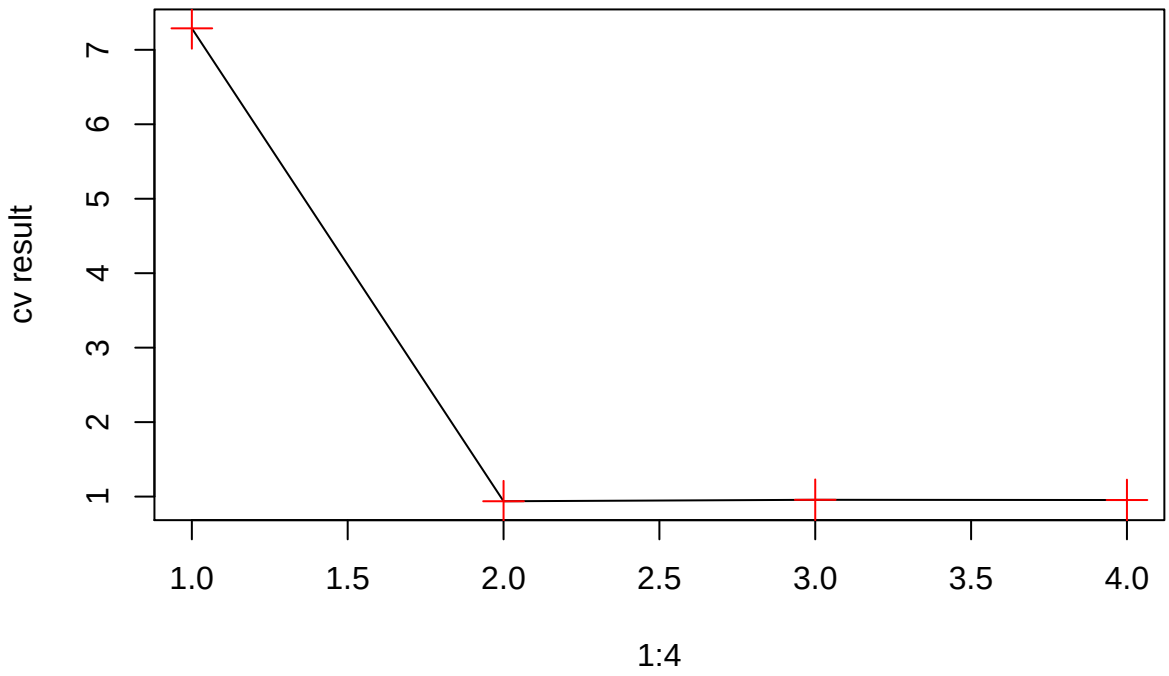
We've generated data from a quadratic Gaussian model. Now forget it is quadratic and let's see if we arrive at the right conclusions using model selection tools.

```
set.seed(42)
```

We'll use LOOCV as a model selection tool for the following models:

```
library(boot)
model1 <- glm(Y~X,data = data)
model2 <- glm(Y~poly(X,2), data = data)
model3 <- glm(Y~poly(X,3), data = data)
model4 <- glm(Y~poly(X,4), data = data)
cv.model1 <- cv.glm(data, glmfit = model1)
cv.model2 <- cv.glm(data, glmfit = model2)
cv.model3 <- cv.glm(data, glmfit = model3)
cv.model4 <- cv.glm(data, glmfit = model4)
```

```
cv.res <- c(cv.model1$delta[1],
       cv.model2$delta[1],
       cv.model3$delta[1],
       cv.model4$delta[1])
plot(1:4, cv.res, type="l", ylab="cv result")
points(1:4,cv.res, pch=3, col=2, cex=2)
```




```
which.min(cv.res)
```

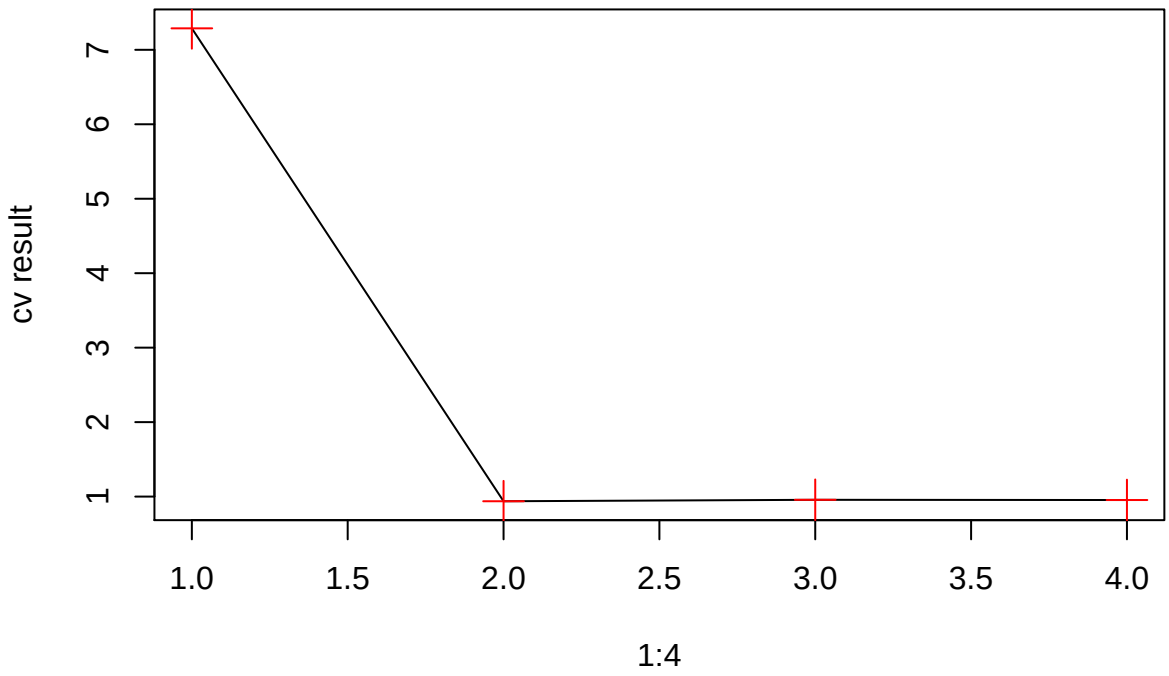
```
## [1] 2
```

The LOOCV confirms that the best model is the second order model. Let's use another random seed to see if we get different results.

```
set.seed(30)
```

```
set.seed(30)
model1 <- glm(Y~X,data = data)
model2 <- glm(Y~poly(X,2), data = data)
model3 <- glm(Y~poly(X,3), data = data)
model4 <- glm(Y~poly(X,4), data = data)
cv.model1 <- cv.glm(data, glmfit = model1)
cv.model2 <- cv.glm(data, glmfit = model2)
cv.model3 <- cv.glm(data, glmfit = model3)
cv.model4 <- cv.glm(data, glmfit = model4)
```

```
cv.res <- c(cv.model1$delta[1],
       cv.model2$delta[1],
       cv.model3$delta[1],
       cv.model4$delta[1])
plot(1:4, cv.res, type="l", ylab="cv result")
points(1:4,cv.res, pch=3, col=2, cex=2)
```




```
which.min(cv.res)
```

```
## [1] 2
```

As expected we get the same exact results:the reason for this is that the LOOCV has no variance due to splittings, hence, repeating the process on the same dataset would always yield the same results.

Let's have a look at the summaries for these model to evaluate the significance of the variables and compare the results of this analysis with the LOOCV results.

```
summary(model1)
```

```
##
## Call:
## glm(formula = Y ~ X, data = data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -9.5161  -0.6800   0.6812   1.5491   3.8183
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.6254     0.2619  -6.205 1.31e-08 ***
## X             0.6925     0.2909   2.380   0.0192 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 6.760719)
```

```
##
##     Null deviance: 700.85  on 99  degrees of freedom
## Residual deviance: 662.55  on 98  degrees of freedom
## AIC: 478.88
##
## Number of Fisher Scoring iterations: 2
```

```
summary(model2)
```

```
##
## Call:
## glm(formula = Y ~ poly(X, 2), data = data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9650  -0.6254  -0.1288   0.5803   2.2700
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.5500     0.0958  -16.18  < 2e-16 ***
## poly(X, 2)1   6.1888     0.9580    6.46 4.18e-09 ***
## poly(X, 2)2 -23.9483     0.9580  -25.00  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.9178258)
##
##     Null deviance: 700.852  on 99  degrees of freedom
## Residual deviance:  89.029  on 97  degrees of freedom
## AIC: 280.17
##
## Number of Fisher Scoring iterations: 2
```

```
summary(model3)
```

```
##
## Call:
## glm(formula = Y ~ poly(X, 3), data = data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9765  -0.6302  -0.1227   0.5545   2.2843
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.55002    0.09626 -16.102  < 2e-16 ***
## poly(X, 3)1   6.18883    0.96263   6.429 4.97e-09 ***
## poly(X, 3)2 -23.94830    0.96263 -24.878  < 2e-16 ***
## poly(X, 3)3   0.26411    0.96263   0.274    0.784
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.9266599)
##
##     Null deviance: 700.852  on 99  degrees of freedom
```

```
## Residual deviance:  88.959  on 96  degrees of freedom
## AIC: 282.09
##
## Number of Fisher Scoring iterations: 2
```

```
summary(model4)
```

```
##
## Call:
## glm(formula = Y ~ poly(X, 4), data = data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0550  -0.6212  -0.1567   0.5952   2.2267
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.55002    0.09591 -16.162  < 2e-16 ***
## poly(X, 4)1   6.18883    0.95905   6.453 4.59e-09 ***
## poly(X, 4)2 -23.94830    0.95905 -24.971  < 2e-16 ***
## poly(X, 4)3   0.26411    0.95905   0.275    0.784
## poly(X, 4)4   1.25710    0.95905   1.311    0.193
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.9197797)
##
##     Null deviance: 700.852  on 99  degrees of freedom
## Residual deviance:  87.379  on 95  degrees of freedom
## AIC: 282.3
##
## Number of Fisher Scoring iterations: 2
```

Now let's look at the anova:

```
anova(model1,model2,model3,model4)
```

```
## Analysis of Deviance Table
##
## Model 1: Y ~ X
## Model 2: Y ~ poly(X, 2)
## Model 3: Y ~ poly(X, 3)
## Model 4: Y ~ poly(X, 4)
##   Resid. Df Resid. Dev Df Deviance
## 1        98     662.55
## 2        97      89.03  1   573.52
## 3        96      88.96  1     0.07
## 4        95      87.38  1     1.58
```

All the tests results above seem to agree with the LOOCV: the third and fourth order terms are superflous.