Statistical_Learning_HW1

2022-03-19

Exercise 1

Assume that

$$Y = f(X) + \epsilon,\ f(X) = e^{-8\|X\|_2^2}$$

with $\epsilon \sim N(0, \sigma^2)$, X is independent and uniformly distributed on $[-1, 1]^p$ and

$$\|X\|_2^2 = \sum_{i=1}^{p} X_i^2$$

corresponds to the squared Euclidean norm of the vector X.

- Use simulations to approximate the expected prediction error (EPE) given by

  $$EPE_{\hat{f}_\tau}(x_0) = E[(Y - \hat{f}_\tau(x_0))^2 | X = x_0] = Var(Y|X = x_0) + [Bias_\tau^2(\hat{f}_\tau(x_0)) + Var_\tau(\hat{f}_\tau(x_0))]$$

  with τ the training sample at $x_0 = 0$ based on $m = 1000$ repetitions.

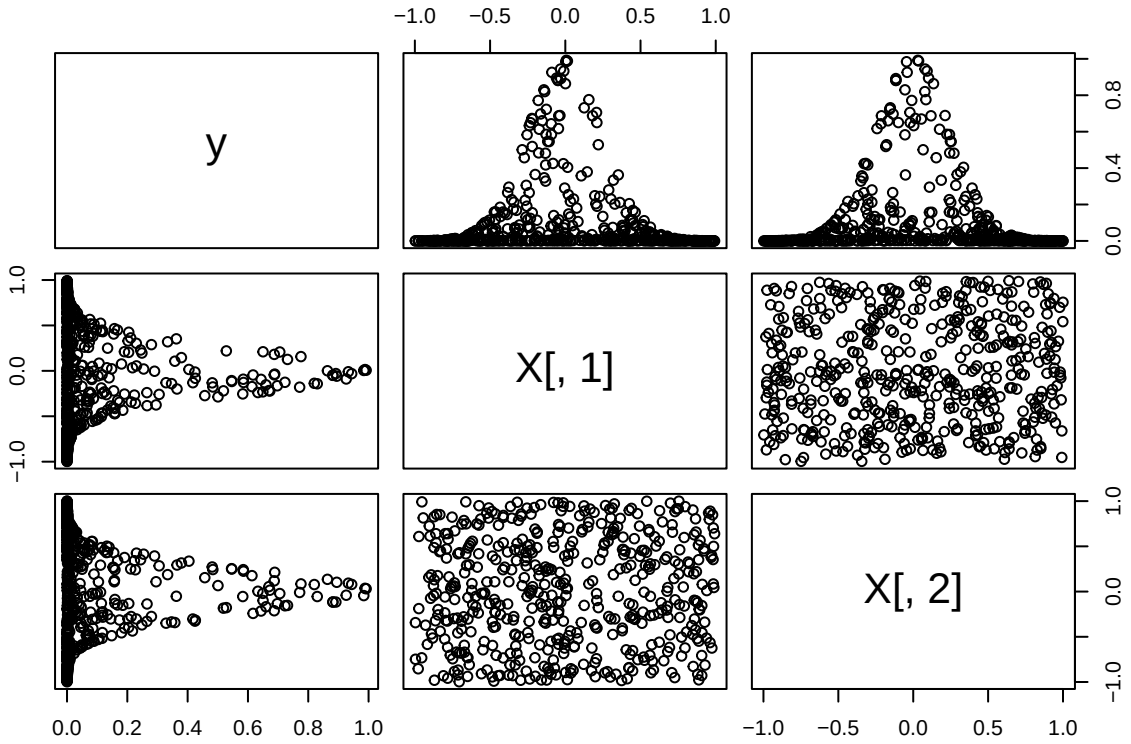
  - for a training sample size of $N = 500$,
  - the number of dimensions p varying from 1 to 10,
  - with a standard deviation σ of either zero or one,
  - using linear models (with an intercept) as well as 1-nearest neighbors to estimate $f(X)$.

- Assess and interpret the impact of the dimension on the EPE in dependence of the method used as well as the value of the standard deviation.

Let us first visualize the problem for $p = 2$ and $\sigma = 0$. This means that we observe data without irreducable noise ϵ. Note further, that in the given example the expected prediction error at x_0 is equal to the mean squared error since the problem is deterministic. This means that the EPE allows for a bias-variance decomposition.

```
library(uniformly)
library(FNN)

euclidean <- function(a, b){
  vector <- numeric(nrow(b))
  for(i in 1:nrow(b)){
    vector[i] <- sqrt(sum((a - b[i,])^2))
  }
  return(vector)
}

onenn <- function(target,x){
```

```
  neighbor <- which.min(euclidean(target,x))
  #return(func(x[neighbor,])+rnorm(1,0,sigma))
  return(neighbor)
}

f <- function(x){
  exp(-8*sum(x^2))
}


X <-runif_in_cube(500,2)
y <- numeric(500)
for (i in 1:500){
  y[i] <- f(X[i,])+rnorm(1,0,0)
}
pairs(y~X[,1]+X[,2])
```

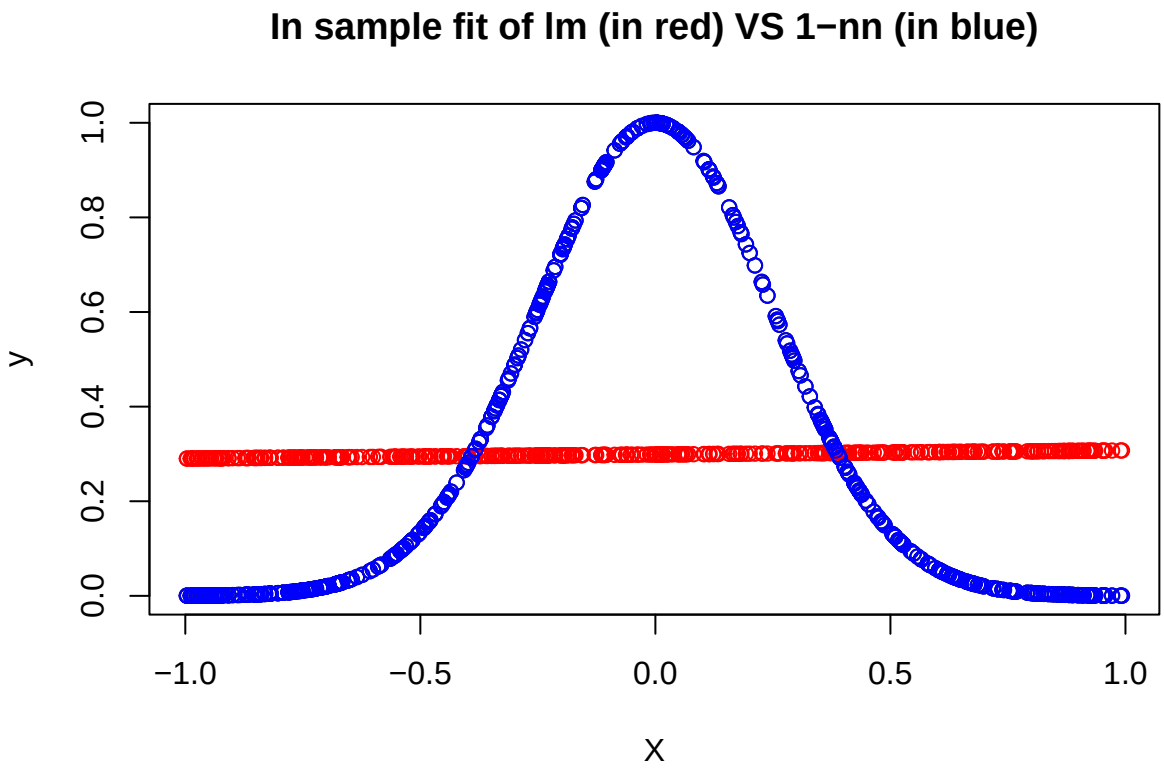


We clearly see that due to the Gaussia kernel $e^{-8\|X\|_2^2}$ we observe the typical bell-shaped form. The conjecture is that due to this non-linear form the 1-nearest neighbor method will give a better in sample fit.

Let us take a look at the in sample fit for dimension 1 and standard deviation 0.

```
X <-runif_in_cube(500,1)
y <- numeric(500)
for (i in 1:500){
  y[i] <- f(X[i,])+rnorm(1,0,0)
}
```

```
y_hat_model <- lm(y~X)
y_hat <- predict(y_hat_model)
#y_hat_2_model <- knn.reg(train=X,y=y,k=1)
#y_hat_2 <- y_hat_2_model$pred
y_hat_2 <- numeric(500)
for(i in 1:500){
  y_hat_2[i] <- y[onenn(X[i,],X)]
}
plot(X,y,main="In sample fit of lm (in red) VS 1-nn (in blue)")
points(X,y_hat,col="red")
points(X,y_hat_2,col="blue")
```

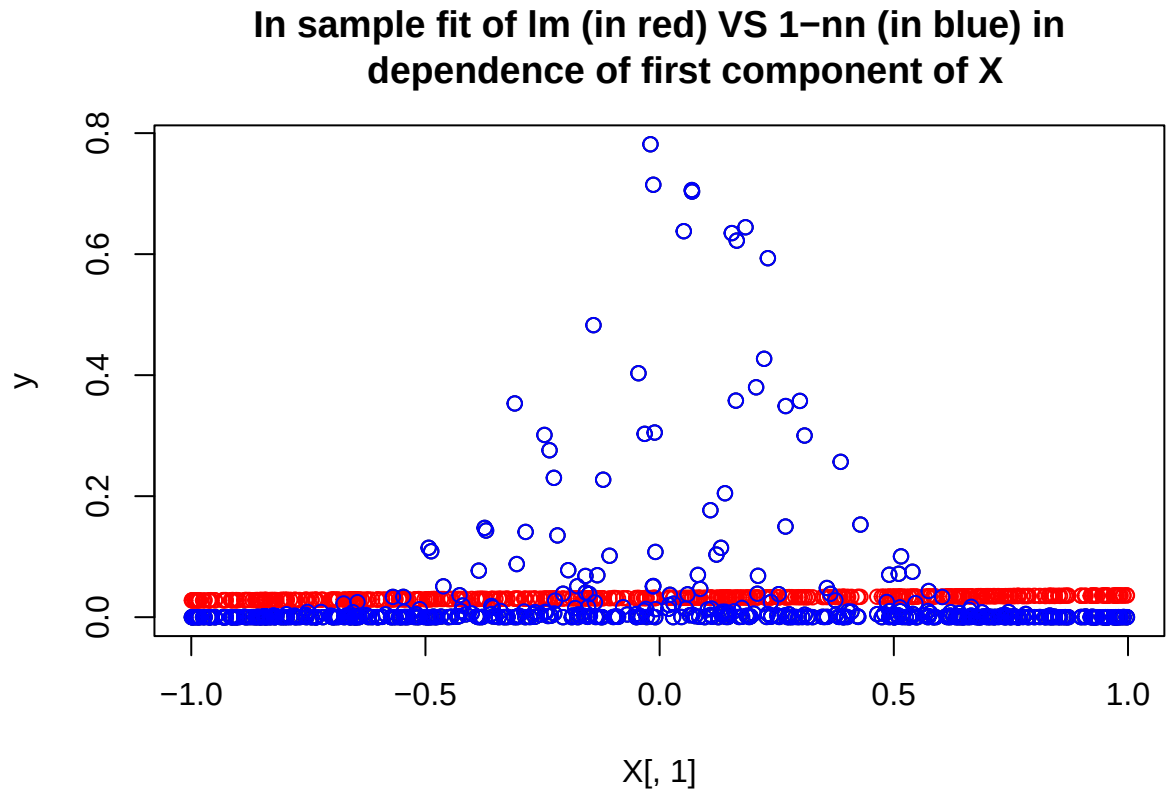


Clearly the 1-nn method gives a better fit.

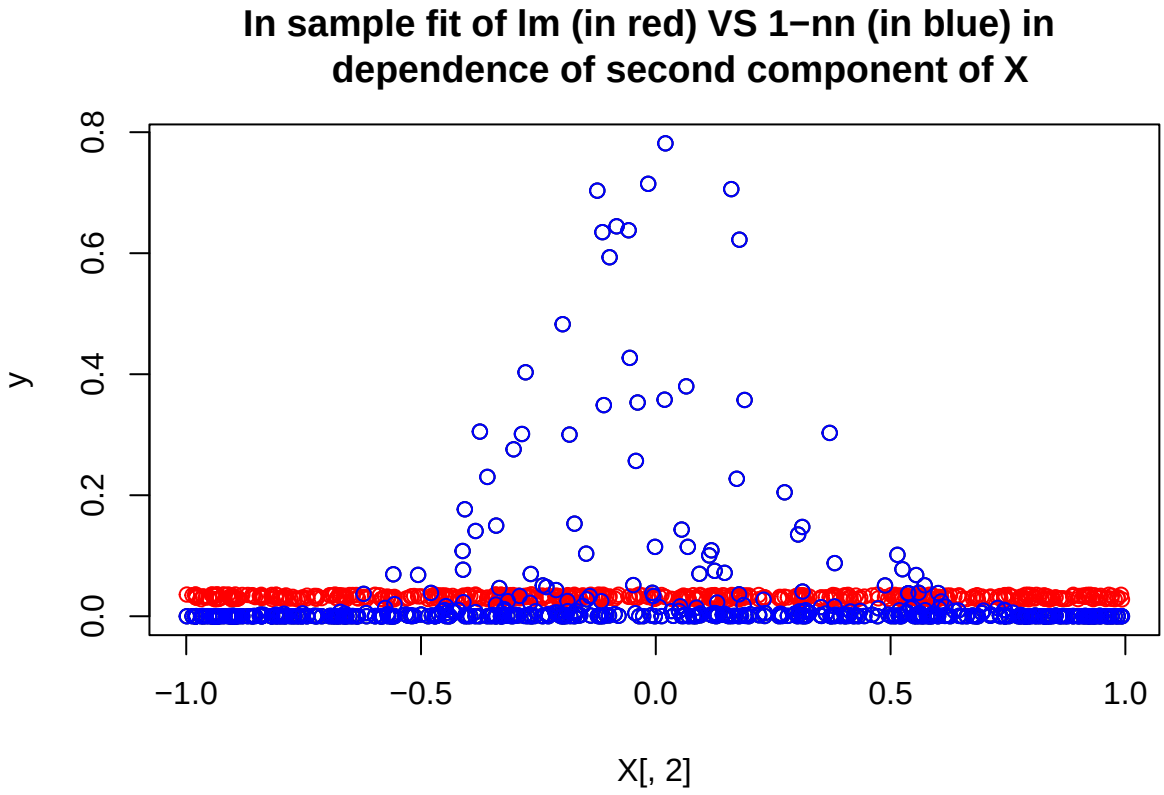
Let us take a look at the in sample fit for dimension 3 and standard deviation 0.

```
X <-runif_in_cube(500,3)
y <- numeric(500)
for (i in 1:500){
  y[i] <- f(X[i,])+rnorm(1,0,0)
}

y_hat_model <- lm(y~X)
y_hat <- predict(y_hat_model)
#y_hat_2_model <- knn.reg(train=X,y=y,k=1)
#y_hat_2 <- y_hat_2_model$pred
```

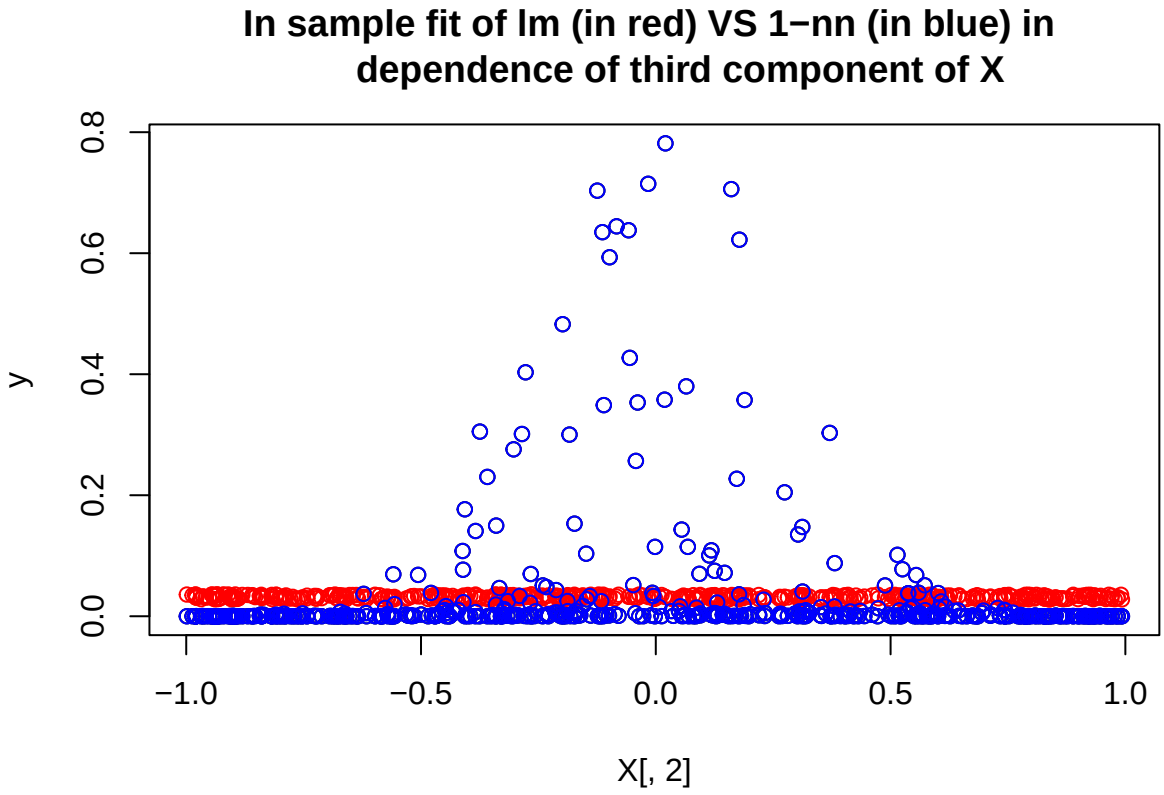
```
y_hat_2 <- numeric(500)
for(i in 1:500){
  y_hat_2[i] <- y[onenn(X[i,],X)]
}
plot(X[,1],y,main="In sample fit of lm (in red) VS 1-nn (in blue) in
     dependence of first component of X")
points(X[,1],y_hat,col="red")
points(X[,1],y_hat_2,col="blue")
```




```
plot(X[,2],y,main="In sample fit of lm (in red) VS 1-nn (in blue) in
     dependence of second component of X")
points(X[,2],y_hat,col="red")
points(X[,2],y_hat_2,col="blue")
```




```
plot(X[,2],y,main="In sample fit of lm (in red) VS 1-nn (in blue) in
     dependence of third component of X")
points(X[,2],y_hat,col="red")
points(X[,2],y_hat_2,col="blue")
```



Again we see that of course the 1-nn method gives a perfect in sample fit.

```
f <- function(x){
  exp(-8*sum(x^2))
}

EPE <- function(p,sigma,method="lm"){
  X <-runif_in_cube(500,p)
  y <- numeric(500)
  for (i in 1:500){
    y[i] <- f(X[i,])+rnorm(1,0,sigma)
  }
  if(method=="lm"){
    y_hat_model <- lm(y~X)
    y_hat <- y_hat_model$coefficients[1]
    return(1-y_hat)
  }
  if(method=="1-nn"){
    #y_hat <- knn.reg(train=rbind(X,c(rep(0,p))),y=y,k=1)
    y_hat <- y[onenn(rep(0,p),X)]
    return(1-y_hat)
  }
  else{
    print("Method not known")
  }
}
```

```r
EPE_repeat <- function(rep,p,sigma,method){
  EPE_vector <- numeric(rep)
  for(k in 1:rep){
    EPE_vector[k] <- EPE(p,sigma,method)
  }
  return(c(mean(EPE_vector)^2,var(EPE_vector)))
}

res_list <- list()

m <- "lm"
matrix <- matrix(nrow=10,ncol=2)
matrix2 <- matrix(nrow=10,ncol=2)
matrix3 <- matrix(nrow=10,ncol=2)
sig <- c(0,1)
for(k in 1:2){
  sigma <- sig[k]
  for(p in 1:10){
    #print(paste("p = ",p,",sigma = ",sigma, ",method = ",m, ",EPE = ",EPE_repeat(1000,p,sigma,m)))
    result <- EPE_repeat(1000,p,sigma,m)
    matrix[p,k] <- sum(result)+sigma
    matrix2[p,k] <-result[1]
    matrix3[p,k] <-result[2]
  }

}
rownames(matrix) <-  paste("p",c(1:10))
colnames(matrix) <- paste("sigma",c(0:1))
res_list[[1]] <- matrix
res_list[[3]] <- matrix2
res_list[[5]] <- matrix3

m <- "1-nn"
matrix <- matrix(nrow=10,ncol=2)
matrix2 <- matrix(nrow=10,ncol=2)
matrix3 <- matrix(nrow=10,ncol=2)
sig <- c(0,1)
for(k in 1:2){
  sigma <- sig[k]
  for(p in 1:10){
    #print(paste("p = ",p,",sigma = ",sigma, ",method = ",m, ",EPE = ",EPE_repeat(1000,p,sigma,m)))
    result <- EPE_repeat(1000,p,sigma,m)
    matrix[p,k] <- sum(result)+sigma
    matrix2[p,k] <-result[1]
    matrix3[p,k] <-result[2]
  }

}
rownames(matrix) <-  paste("p",c(1:10))
colnames(matrix) <- paste("sigma",c(0:1))
res_list[[2]] <- matrix
res_list[[4]] <- matrix2
res_list[[6]] <- matrix3
```

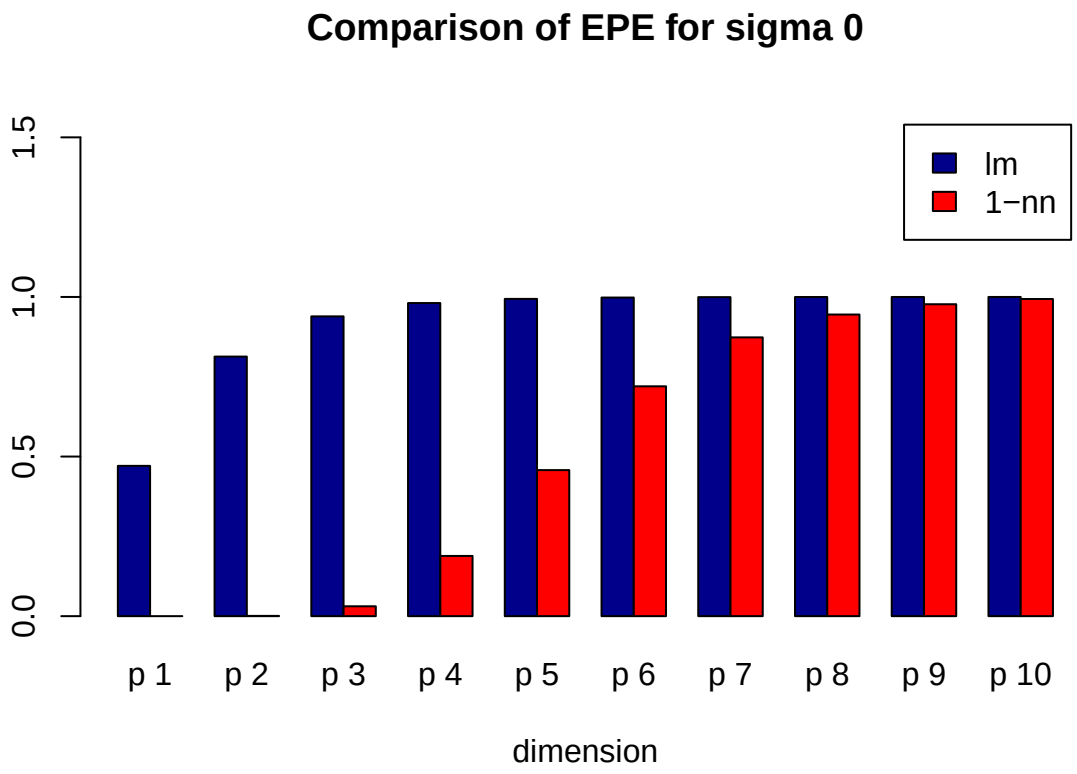
```
kable(res_list[[1]], digits = 5)
```

	sigma 0	sigma 1
p 1	0.47085	1.47230
p 2	0.81323	1.81953
p 3	0.93919	1.94316
p 4	0.98098	1.98437
p 5	0.99394	1.99468
p 6	0.99814	1.99841
p 7	0.99941	2.00094
p 8	0.99981	2.00331
p 9	0.99994	2.00295
p 10	0.99998	2.00058

```
kable(res_list[[2]], digits = 5)
```

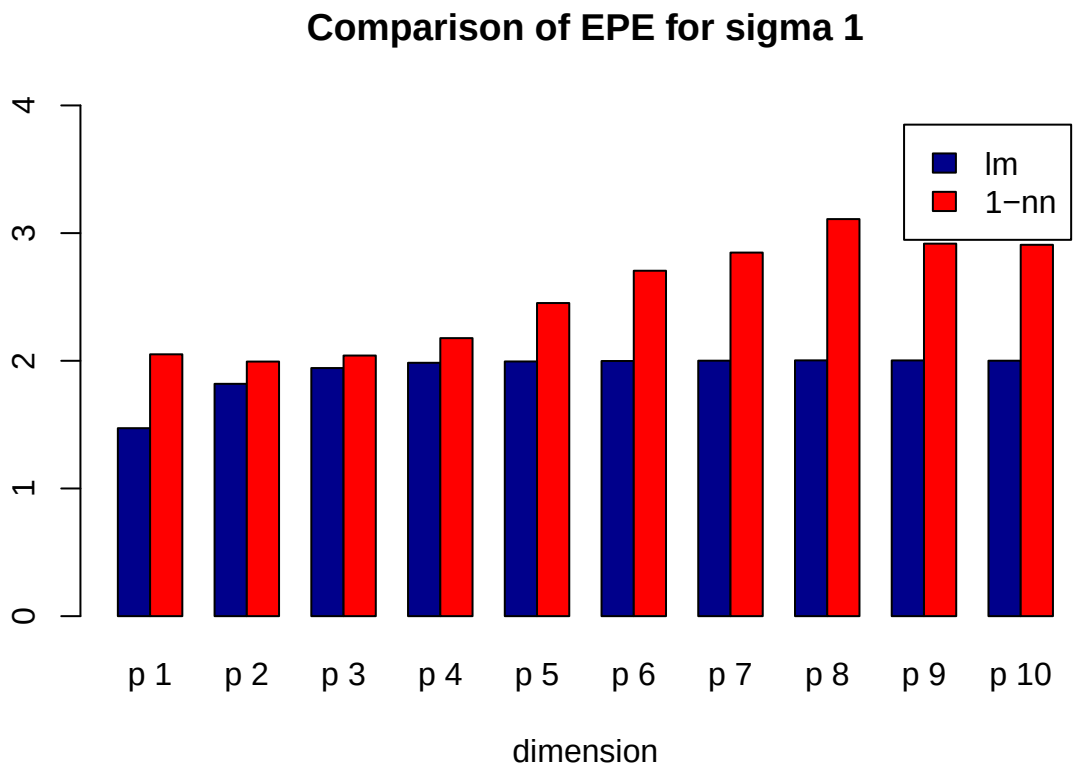
	sigma 0	sigma 1
p 1	0.00000	2.05073
p 2	0.00080	1.99400
p 3	0.03088	2.04109
p 4	0.18866	2.17745
p 5	0.45757	2.45244
p 6	0.72007	2.70524
p 7	0.87333	2.84752
p 8	0.94495	3.10980
p 9	0.97716	2.91753
p 10	0.99354	2.90842

```
barplot(rbind(res_list[[1]][,1],res_list[[2]][,1]),beside=TRUE,main="Comparison of EPE for sigma 0",
        xlab="dimension", col=c("darkblue","red"),
        legend = c("lm","1-nn"),ylim=c(0,1.6))
```



We observe that for $\sigma = 0$ the EPE is lower for the one nearest neighbor method and grows towards 1 as the number of dimensions grows for both methods.

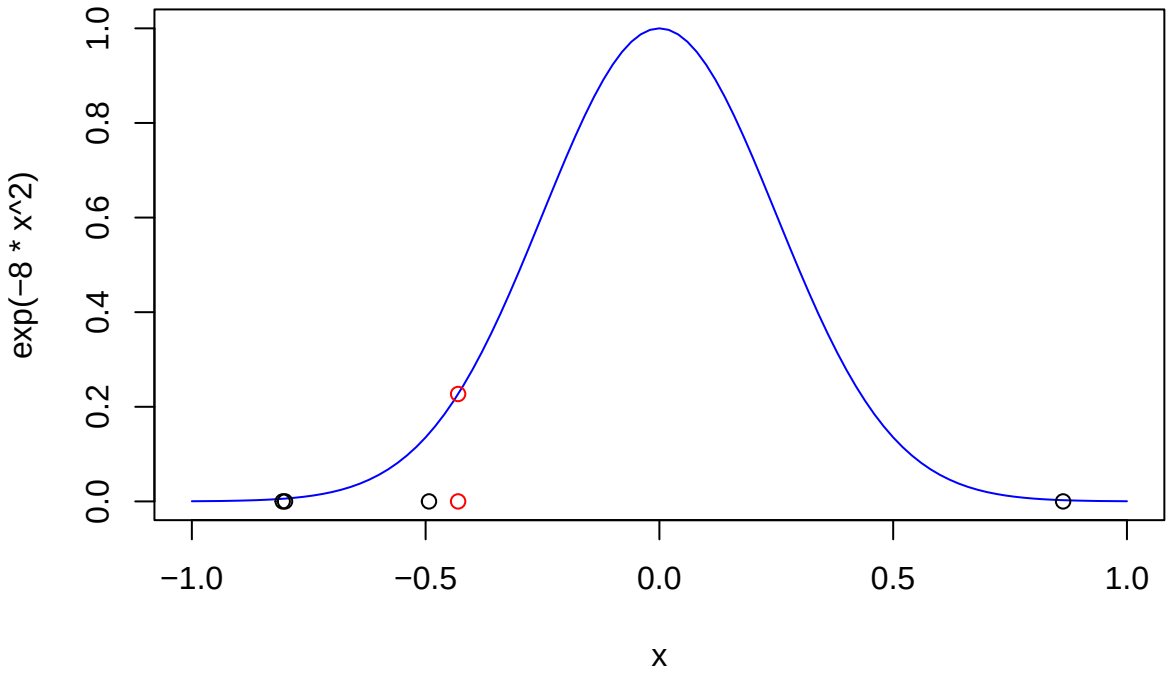
```
barplot(rbind(res_list[[1]][,2],res_list[[2]][,2]),beside=TRUE,main="Comparison of EPE for sigma 1",
        xlab="dimension", col=c("darkblue","red"),
        legend = c("lm","1-nn"),ylim=c(0,4))
```



We observe that for $\sigma = 1$ the EPE seems to be higher for the 1-nn method and increases with p for both methods. It is also immediate that the EPE for the linear model is exactly increased by $\sigma = 1$ in comparison to the previous picture, since while the squared bias still goes to one as p increases the variance is still close to zero, so the term that is added in the EPE is $Var(\epsilon)$.

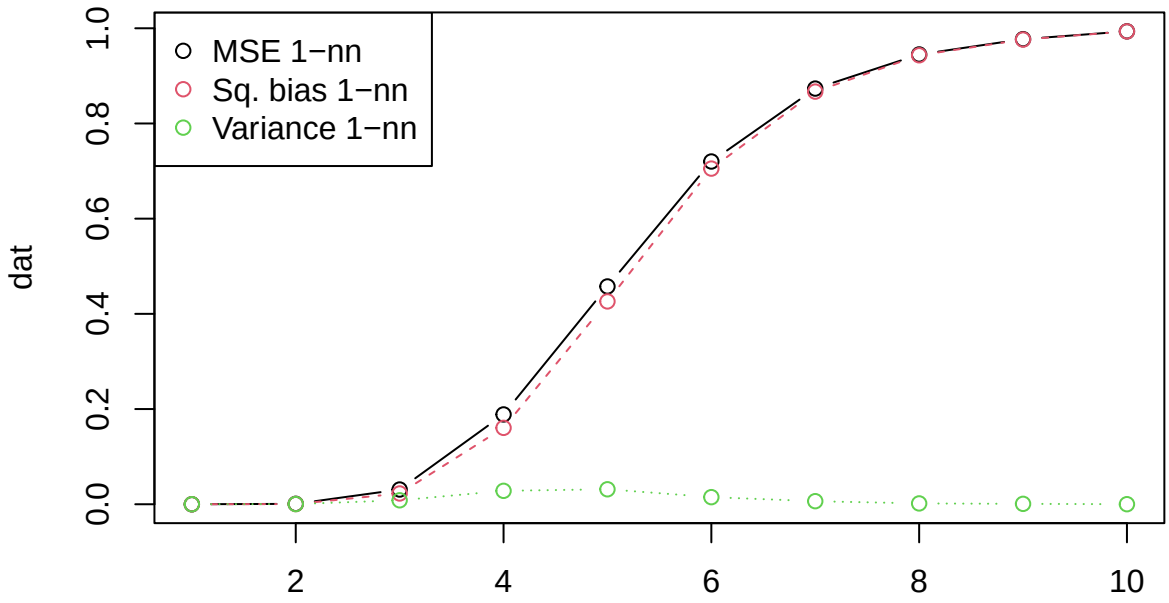
Let us study the results related to the 1-nearest neighbor method more closely in the case where $\sigma = 0$. Except for the case where the nearest neighbor is at 0, $\hat{y}_0$ will be smaller than $f(0)$ in this example, since $f(x)$ attains its maximum at $x_0 = \mathbf{0}$. This results in a downward bias in the average estimate. We illustrate this effect graphically. The 1-nearest neighbor estimate is given by the red dot on the blue curve. The black dots on the x-axis represent five random draws in [-1,1].

```
X_unif <- runif(5,-1,1)
closest <- which.min(abs(0-X_unif))
curve(exp(-8*x^2),from=-1,to=1,col="blue")
points( X_unif[closest],0,col="red")
for(i in X_unif[-closest]){
  points(i,0,col="black")
}
points(X_unif[closest],exp(-8*X_unif[closest]^2),col="red")
```



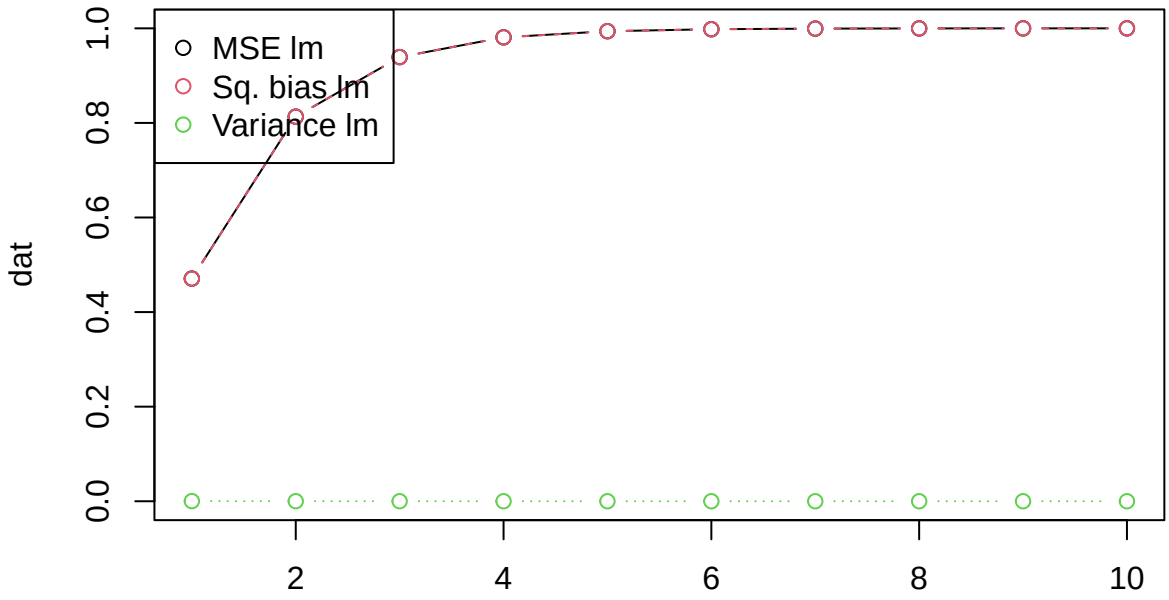
In low dimensions, the nearest neighbor is very close to 0, and so both the bias and variance are small. As the dimension increases, the nearest neighbor tends to stray further from the target point, and both bias and variance are incurred. In this specific example as p increases, the estimate tends to be 0 more often than not, and hence the MSE levels off at 1.0, as does the bias, and the variance starts dropping. This exemplifies the so called curse of dimensionality. And is illustrated below. Note that in this example the bias is the dominant component of the MSE.

```
dat <- matrix(cbind(res_list[[2]][,1],res_list[[4]][,1],res_list[[6]][,1]),ncol=3) # make data
matplot(dat, type = c("b"),pch=1,col = 1:3) #plot
legend("topleft", legend = c("MSE 1-nn","Sq. bias 1-nn","Variance 1-nn"), col=1:3, pch=1)
```




```
# optional legend
```

```
dat <- matrix(cbind(res_list[[1]][,1],res_list[[3]][,1],res_list[[5]][,1]),ncol=3) # make data
matplot(dat, type = c("b"),pch=1,col = 1:3) #plot
legend("topleft", legend = c("MSE lm","Sq. bias lm","Variance lm"), col=1:3, pch=1) # optional legend
```



Overall it is interesting to see that even if the linear regression method fails to capture the true shape of the function, the EPE is approximately the same for both methods. This means that the error the linear regression method makes 'averages out'. (high bias, low variance)

Exercise 2

Assume that

$$Y = f(X) + \epsilon,\ f(X) = X_1$$

with $\epsilon \sim N(0, \sigma^2)$, X is independent and uniformly distributed on $[-1, 1]^p$. X_1 denotes the first element of the p-dimensional vector X.

- Use simulations to approximate the expected prediction error (EPE) given by

  $$EPE_{\hat{f}_\tau}(x_0) = E[(Y - \hat{f}_\tau(x_0))^2 | X = x_0] = Var(Y|X = x_0) + [Bias^2_\tau(\hat{f}_\tau(x_0)) + Var_\tau(\hat{f}_\tau(x_0))]$$

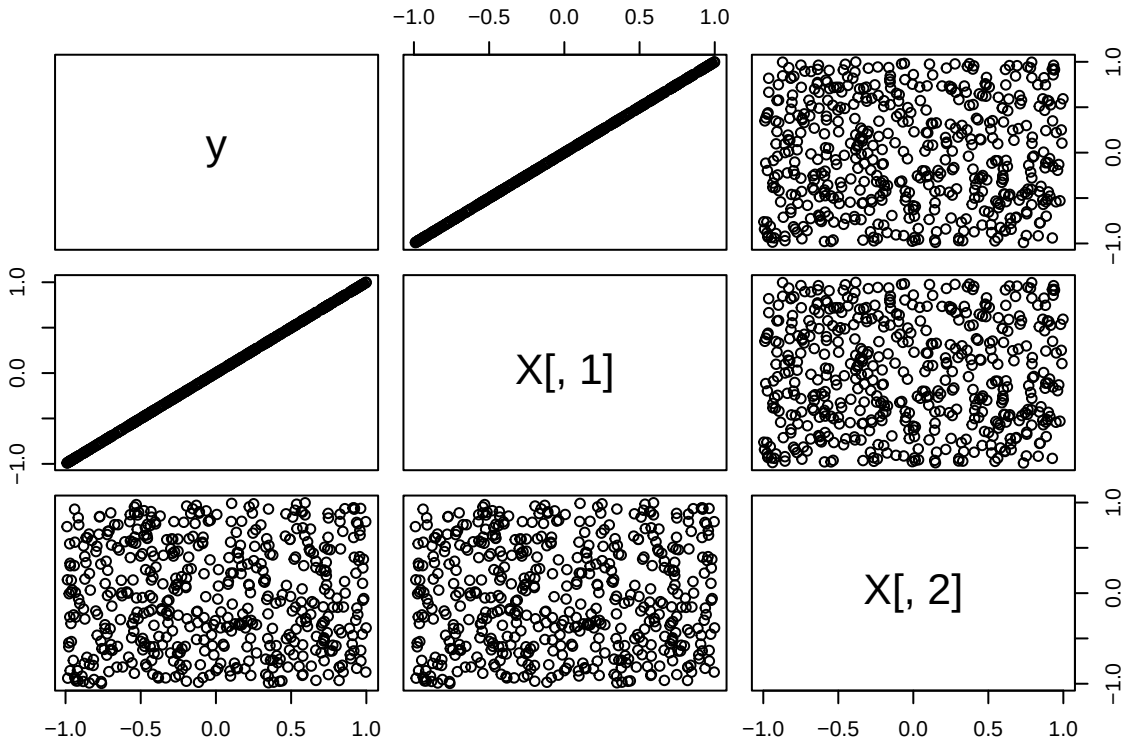
  with τ the training sample at $x_0 = 0$ based on $m = 1000$ repetitions.

  - for a training sample size of $N = 500$,
  - the number of dimensions p varying from 1 to 10,
  - with a standard deviation σ of either zero or one,
  - using linear models (with an intercept) as well as 1-nearest neighbors to estimate $f(X)$.

- Assess and interpret the impact of the dimension on the EPE in dependence of the method used as well as the value of the standard deviation.

Let us first visualize the problem for $p = 2$ and $\sigma = 0$.

```
f <- function(x){
  x[1]
}


X <-runif_in_cube(500,2)
y <- numeric(500)
for (i in 1:500){
  y[i] <- f(X[i])+rnorm(1,0,0)
}
pairs(y~X[,1]+X[,2])
```

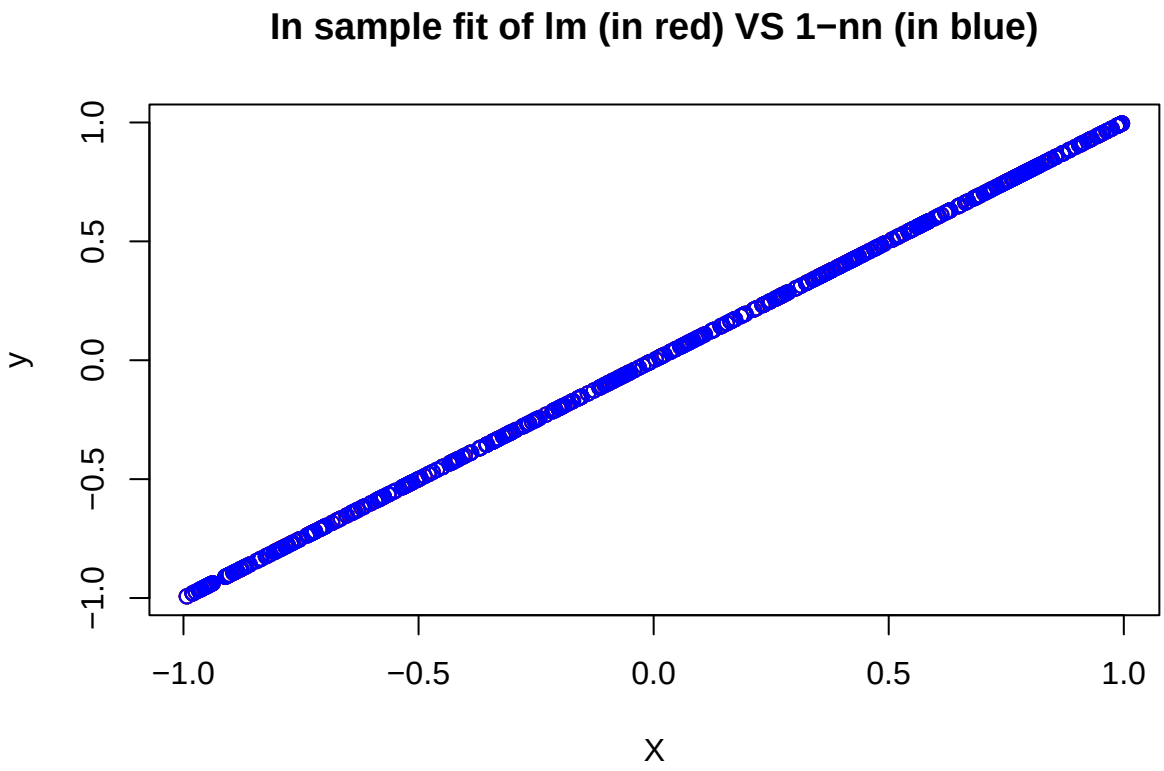


As expected we see a clear linear relationship (identity) in the first component due to the form of $f(x)$. The conjecture is that therefore the linear regression method will perform better once an irreducable error component is introduced.

Let us take a look at the in sample fit for dimension 1 and standard deviation 0.

```
X <-runif_in_cube(500,1)
y <- numeric(500)
for (i in 1:500){
  y[i] <- f(X[i,])+rnorm(1,0,0)
}

y_hat_model <- lm(y~X)
```

```r
y_hat <- predict(y_hat_model)
#y_hat_2_model <- knn.reg(train=X,y=y,k=1)
#y_hat_2 <- y_hat_2_model$pred
y_hat_2 <- numeric(500)
for(i in 1:500){
  y_hat_2[i] <- y[onenn(X[i,],X)]
}
plot(X,y,main="In sample fit of lm (in red) VS 1-nn (in blue)")
points(X,y_hat,col="red")
points(X,y_hat_2,col="blue")
```

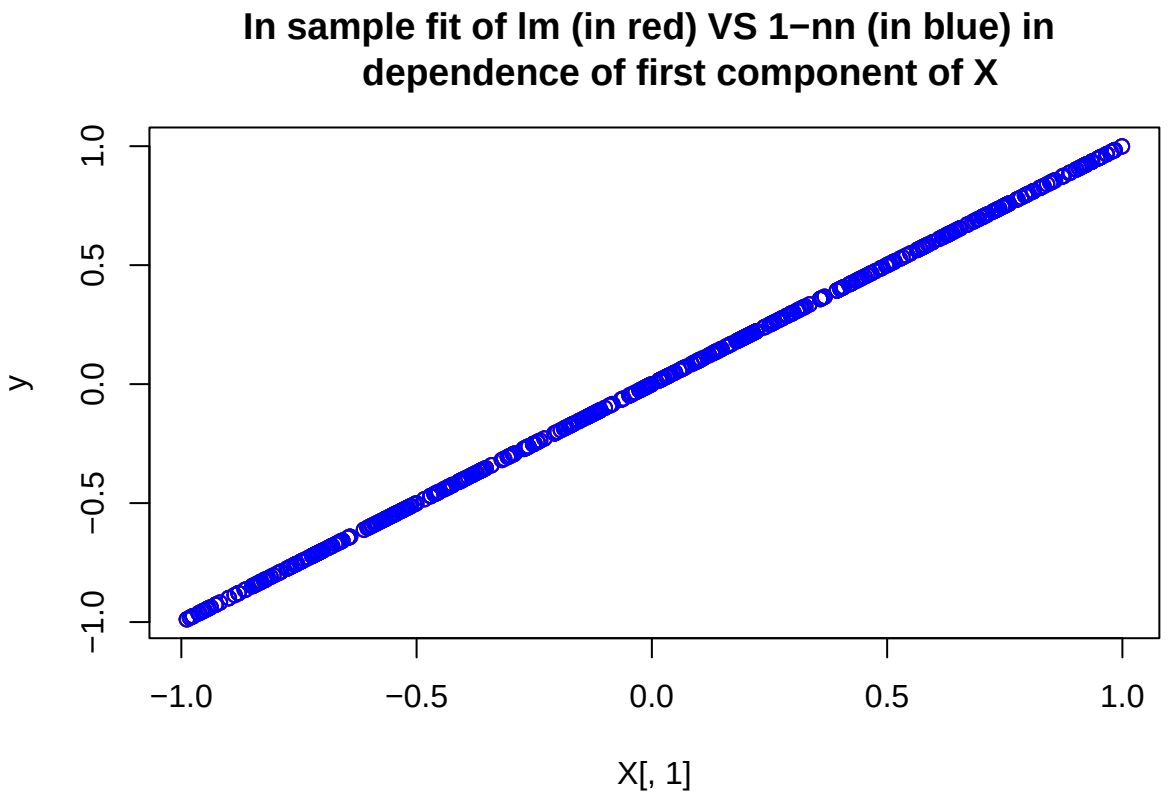


Without irreducably noise both models capture the relationship perfectly. Note that the linear model is unbiased in this example.

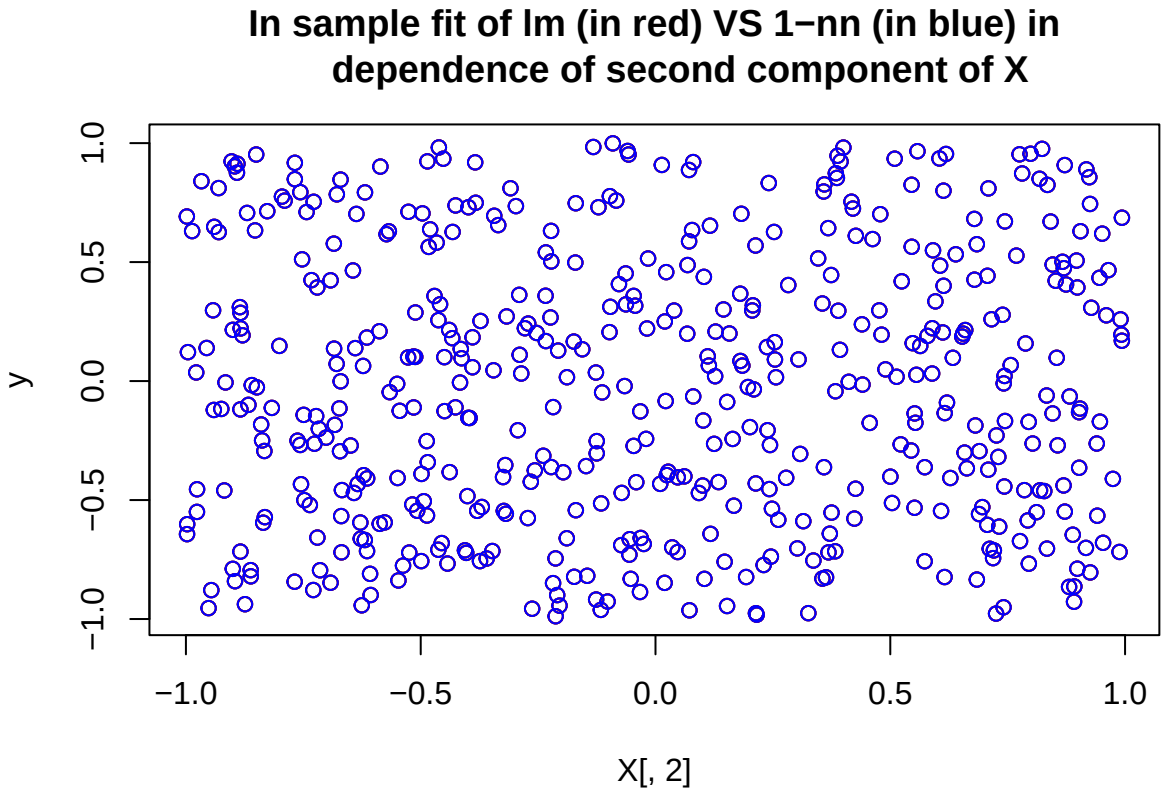
Let us take a look at the in sample fit for dimension 3 and standard deviation 0.

```r
X <-runif_in_cube(500,3)
y <- numeric(500)
for (i in 1:500){
  y[i] <- f(X[i,])+rnorm(1,0,0)
}

y_hat_model <- lm(y~X)
y_hat <- predict(y_hat_model)
#y_hat_2_model <- knn.reg(train=X,y=y,k=1)
#y_hat_2 <- y_hat_2_model$pred
y_hat_2 <- numeric(500)
```

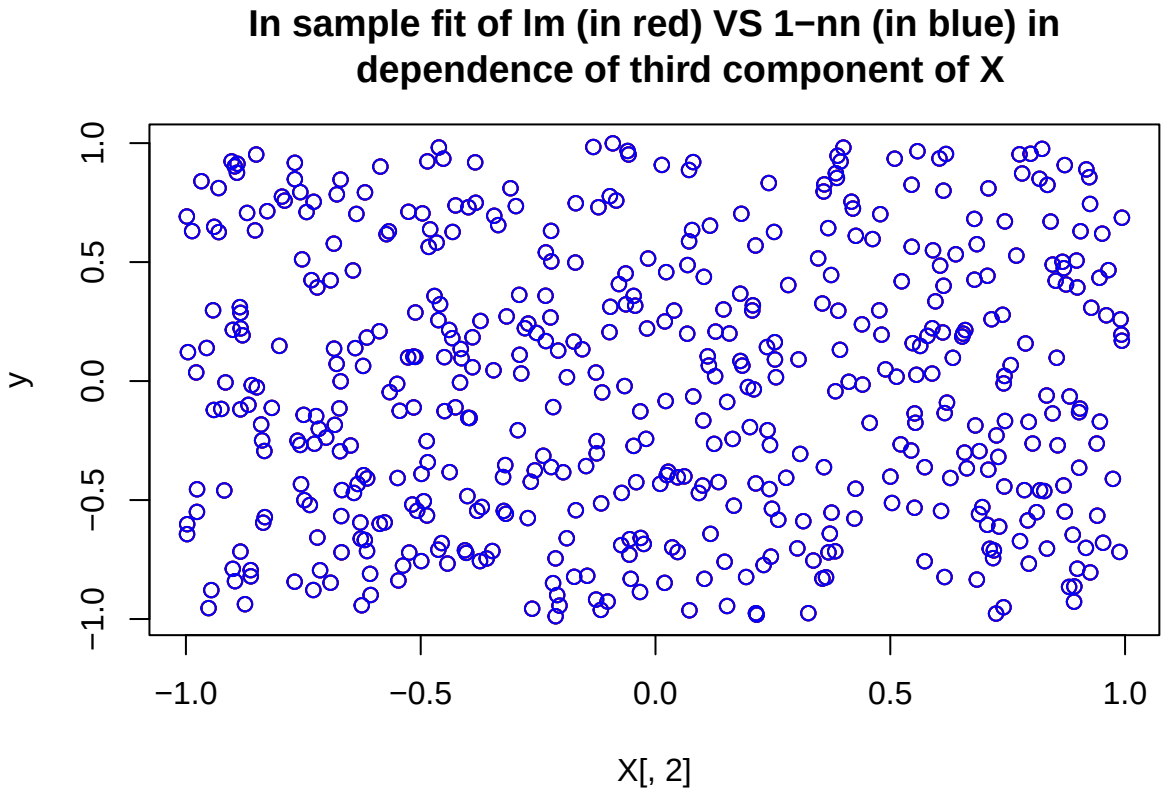
```
for(i in 1:500){
  y_hat_2[i] <- y[onenn(X[i,],X)]
}
plot(X[,1],y,main="In sample fit of lm (in red) VS 1-nn (in blue) in
     dependence of first component of X")
points(X[,1],y_hat,col="red")
points(X[,1],y_hat_2,col="blue")
```




```
plot(X[,2],y,main="In sample fit of lm (in red) VS 1-nn (in blue) in
     dependence of second component of X")
points(X[,2],y_hat,col="red")
points(X[,2],y_hat_2,col="blue")
```




```
plot(X[,2],y,main="In sample fit of lm (in red) VS 1-nn (in blue) in
     dependence of third component of X")
points(X[,2],y_hat,col="red")
points(X[,2],y_hat_2,col="blue")
```



Again we can see that the linear model which is unbiased in this case gives a perfect in sample fit. The 1-nn method again results in the obligatory perfect in sample fit.

```
summary(y_hat_model)
```

```
##
## Call:
## lm(formula = y ~ X)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -8.091e-16 -8.320e-17 -3.140e-17  1.940e-17  1.730e-14
##
## Coefficients:
##               Estimate Std. Error    t value Pr(>|t|)
## (Intercept) -3.476e-17  3.503e-17 -9.920e-01    0.322
## X1           1.000e+00  6.232e-17  1.605e+16   <2e-16 ***
## X2          -5.655e-17  5.981e-17 -9.460e-01    0.345
## X3           7.995e-17  5.963e-17  1.341e+00    0.181
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.824e-16 on 496 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 8.604e+31 on 3 and 496 DF,  p-value: < 2.2e-16
```

The summary of the fitted linear regression model also correctly tells us that only the expected effect of the

first component of X is statistically significant, c.p.

```
f <- function(x){
  x[1]
}

EPE <- function(p,sigma,method="lm"){
  X <-runif_in_cube(500,p)
  y <- numeric(500)
  for (i in 1:500){
    y[i] <- f(X[i])+rnorm(1,0,sigma)
  }
  if(method=="lm"){
    y_hat_model <- lm(y~X)
    y_hat <- y_hat_model$coefficients[1]
    return(0-y_hat)
  }
  if(method=="1-nn"){
    #y_hat <- knn.reg(train=rbind(X,c(rep(0,p))),y=y,k=1)
    y_hat <- y[onenn(rep(0,p),X)]
    return(0-y_hat)
  }
  else{
    print("Method not known")
  }
}

EPE_repeat <- function(rep,p,sigma,method){
  EPE_vector <- numeric(rep)
  for(k in 1:rep){
    EPE_vector[k] <- EPE(p,sigma,method)
  }
  return(c(mean(EPE_vector)^2,var(EPE_vector)))
}

res_list <- list()

m <- "lm"
matrix <- matrix(nrow=10,ncol=2)
matrix2 <- matrix(nrow=10,ncol=2)
matrix3 <- matrix(nrow=10,ncol=2)
sig <- c(0,1)
for(k in 1:2){
  sigma <- sig[k]
  for(p in 1:10){
    #print(paste("p = ",p,",sigma = ",sigma, ",method = ",m, ",EPE = ",EPE_repeat(1000,p,sigma,m)))
    result <- EPE_repeat(1000,p,sigma,m)
    matrix[p,k] <- sum(result)+sigma
    matrix2[p,k] <-result[1]
    matrix3[p,k] <-result[2]
  }

}
rownames(matrix) <-  paste("p",c(1:10))
```

```
colnames(matrix) <- paste("sigma",c(0:1))
res_list[[1]] <- matrix
res_list[[3]] <- matrix2
res_list[[5]] <- matrix3

m <- "1-nn"
matrix <- matrix(nrow=10,ncol=2)
matrix2 <- matrix(nrow=10,ncol=2)
matrix3 <- matrix(nrow=10,ncol=2)
sig <- c(0,1)
for(k in 1:2){
  sigma <- sig[k]
  for(p in 1:10){
    #print(paste("p = ",p,",sigma = ",sigma, ",method = ",m, ",EPE = ",EPE_repeat(1000,p,sigma,m)))
    result <- EPE_repeat(1000,p,sigma,m)
    matrix[p,k] <- sum(result)+sigma
    matrix2[p,k] <-result[1]
    matrix3[p,k] <-result[2]
  }

}
rownames(matrix) <-  paste("p",c(1:10))
colnames(matrix) <- paste("sigma",c(0:1))
res_list[[2]] <- matrix
res_list[[4]] <- matrix2
res_list[[6]] <- matrix3
```

```
kable(res_list[[1]], digits = 5)
```

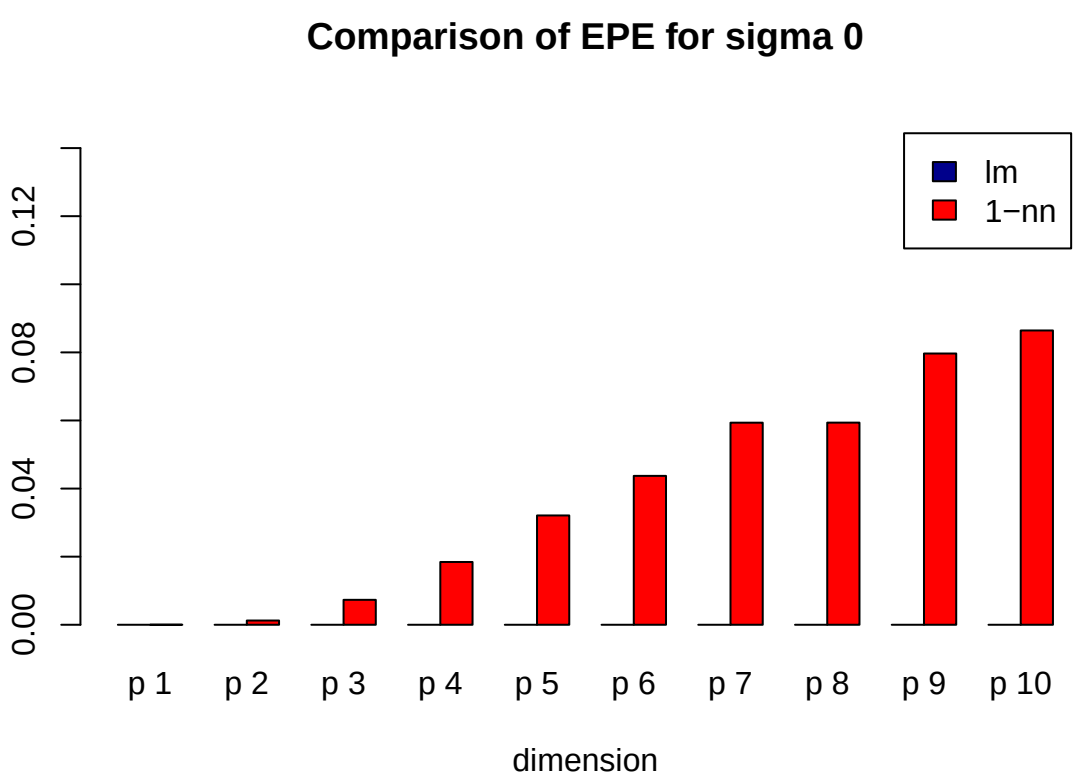
	sigma 0	sigma 1
p 1	0	1.00199
p 2	0	1.00201
p 3	0	1.00205
p 4	0	1.00198
p 5	0	1.00207
p 6	0	1.00194
p 7	0	1.00208
p 8	0	1.00203
p 9	0	1.00212
p 10	0	1.00197

```
kable(res_list[[2]], digits = 5)
```

	sigma 0	sigma 1
p 1	0.00001	2.02127
p 2	0.00124	2.10729
p 3	0.00732	2.00842
p 4	0.01844	2.02904
p 5	0.03210	2.09756

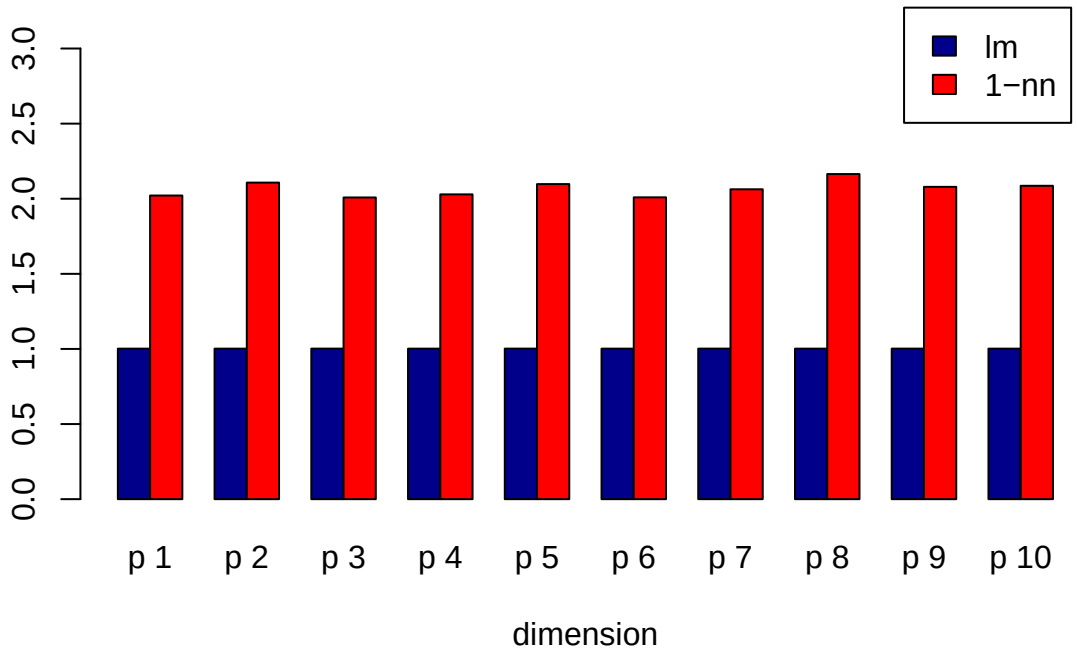
	sigma 0	sigma 1
p 6	0.04372	2.00924
p 7	0.05935	2.06291
p 8	0.05937	2.16419
p 9	0.07966	2.07911
p 10	0.08643	2.08606

```
barplot(rbind(res_list[[1]][,1],res_list[[2]][,1]),beside=TRUE,main="Comparison of EPE for sigma 0",
        xlab="dimension", col=c("darkblue","red"),
        legend = c("lm","1-nn"),ylim=c(0,0.15))
```



As expected we observe that for $\sigma = 0$, the linear regression has an EPE of 0. The EPE related to the 1-nn method is slightly higher and grows with p.

```
barplot(rbind(res_list[[1]][,2],res_list[[2]][,2]),beside=TRUE,main="Comparison of EPE for sigma 1",
        xlab="dimension", col=c("darkblue","red"),
        legend = c("lm","1-nn"),ylim=c(0,3.4))
```



We observe that for $\sigma = 1$ the EPE for the linear regression model is close to 1 for all p. This makes a lot of sense since the irreducable error σ^2 rises by 1, we have unbiasedness and the variance of the estimator is small. The EPE for 1-nearest neighbor is always above 2, since the variance of the estimator in this case is at least $\sigma^2 = 1$, and the ratio increases with dimension as the nearest neighbor strays from the target point However, the curse of dimensionality would suggest that the EPE rises with p. Let us investigate why this is not the case for the linear regression by taking a closer look at the form of the EPE in this case. Since the linear model is unbiased in this scenario the EPE can be written as follows

$$EPE(x_0) = \mathbb{E}_{y_0|x_0}\mathbb{E}_\tau(y_0 - \hat{y}_0)^2 = Var(y_0 \mid x_0) + \mathbb{E}_\tau[\hat{y}_0 - \mathbb{E}_\tau\hat{y}_0]^2 + [\mathbb{E}_\tau\hat{y}_0 - x_0^T\beta]^2$$

$$= Var(\epsilon) + Var_\tau(\hat{y}_0) + Bias^2(\hat{y}_0) = \sigma^2 + \mathbb{E}_\tau(\mathbf{X}^T\mathbf{X})^{-1}x_0\sigma^2 + 0^2$$

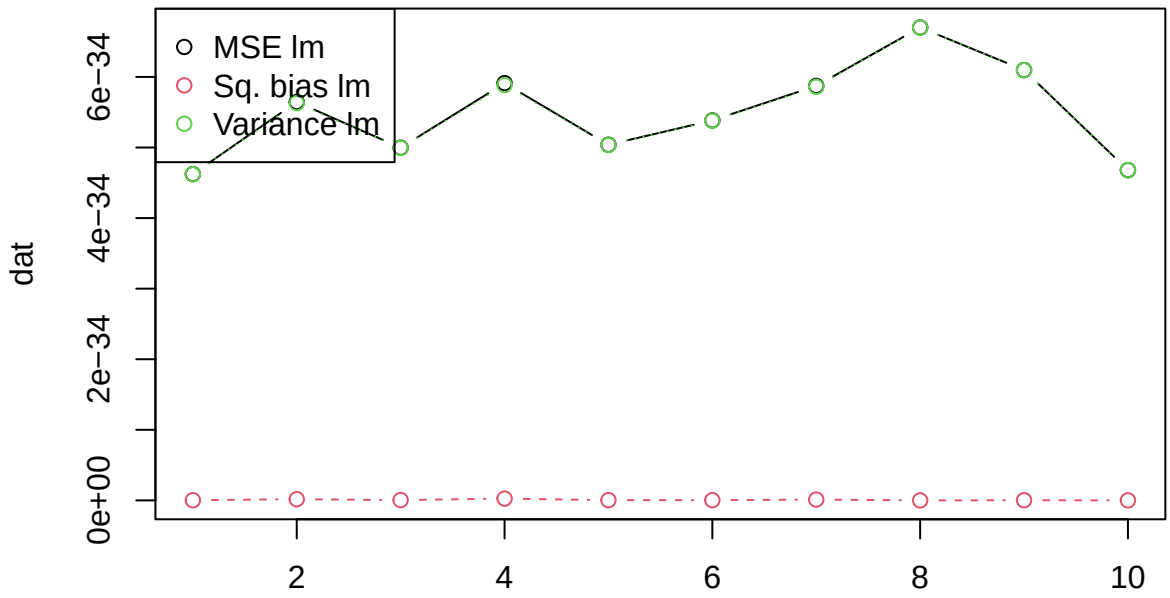
Assume that N is large and τ is selected at random. Since we have that $E(X) = 0$, then $\mathbf{X}^T\mathbf{X} \to NCov(X)$ and

$$\mathbb{E}_{x_0}EPE(x_0) \approx \mathbb{E}_{x_0}x_0^TCov(X)^{-1}x_0\sigma^2/N + \sigma^2$$
$$= trace[Cov(X)^{-1}Cov(x_0)]\sigma^2/N + \sigma^2 = \sigma^2(p/N) + \sigma^2$$

We realize that the EPE is linear in p with slope σ^2/N, meaning that if either N is large or σ small the growth in the variance of the estimator is negligible and zero if $\sigma^2 = 0$. Thus the curse of dimensionality is avoided by construction.

The below plots illustrate once again that in this case the linear model incurs no bias, since the function itself is linear.

```
dat <- matrix(cbind(res_list[[1]][,1],res_list[[3]][,1],res_list[[5]][,1]),ncol=3) # make data
matplot(dat, type = c("b"),pch=1,col = 1:3) #plot
legend("topleft", legend = c("MSE lm","Sq. bias lm","Variance lm"), col=1:3, pch=1) # optional legend
```



Exercise 3

First we load and prepare the data.

```
library(lars)
data("diabetes", package = "lars")
reg.data <- cbind(diabetes$y,diabetes$x)
colnames(reg.data)[1] <- "y"
head(reg.data)
```

```
##        y          age         sex         bmi          map           tc
## [1,] 151  0.038075906  0.05068012  0.06169621  0.021872355 -0.044223498
## [2,]  75 -0.001882017 -0.04464164 -0.05147406 -0.026327835 -0.008448724
## [3,] 141  0.085298906  0.05068012  0.04445121 -0.005670611 -0.045599451
## [4,] 206 -0.089062939 -0.04464164 -0.01159501 -0.036656447  0.012190569
## [5,] 135  0.005383060 -0.04464164 -0.03638469  0.021872355  0.003934852
## [6,]  97 -0.092695478 -0.04464164 -0.04069594 -0.019442093 -0.068990650
##              ldl          hdl          tch          ltg          glu
## [1,] -0.03482076 -0.043400846 -0.002592262  0.019908421 -0.017646125
## [2,] -0.01916334  0.074411564 -0.039493383 -0.068329744 -0.092204050
```

```
## [3,] -0.03419447 -0.032355932 -0.002592262  0.002863771 -0.025930339
## [4,]  0.02499059 -0.036037570  0.034308859  0.022692023 -0.009361911
## [5,]  0.01559614  0.008142084 -0.002592262 -0.031991445 -0.046640874
## [6,] -0.07928784  0.041276824 -0.076394504 -0.041180385 -0.096346157
```

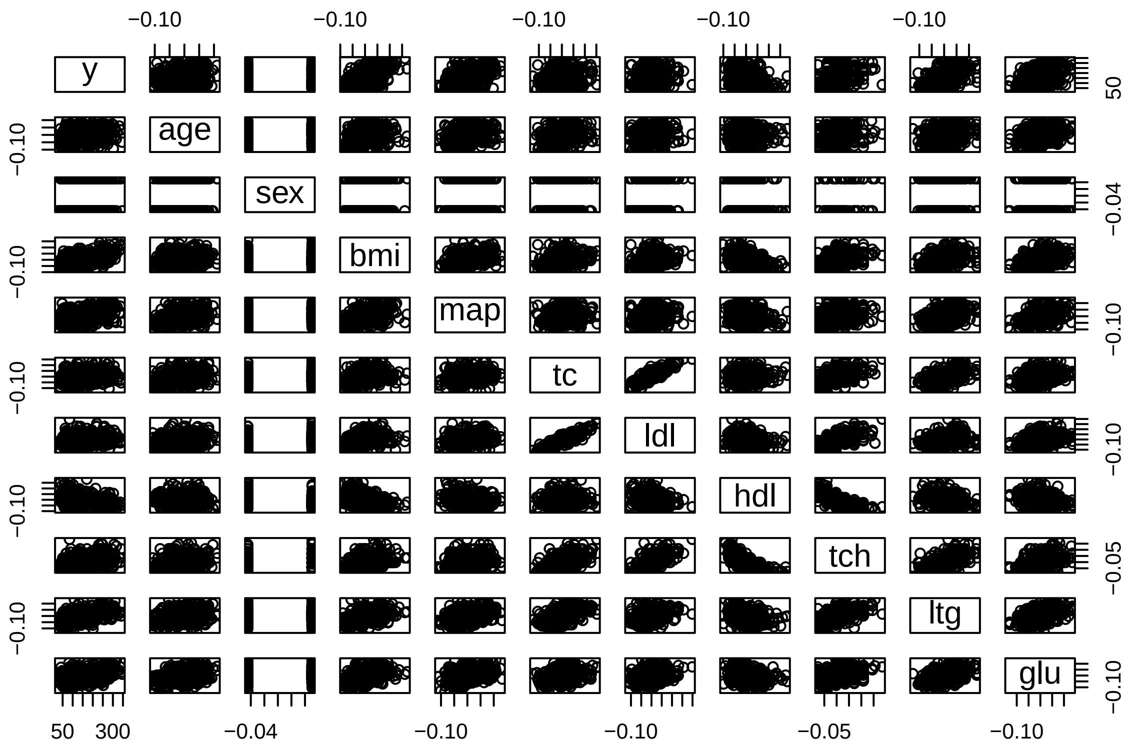
We set a random seed and split the data set into a training and test data set such that 400 observations are used for training and the remaining ones for testing. It makes sense to randomly select 400 observations from the available data set instead of using the first 400, because through random selection we can avoid bias.

```
#splitting randomly into train and test data
set.seed(1)
train.ind <- sample(seq_len(nrow(reg.data)), size = 400)
train <- data.frame(reg.data[train.ind, ])
test <- data.frame(reg.data[-train.ind, ])
```

The covariates are only available in standardized form. Interpreting the regression output correctly can therefore be an issue for the subsequent analysis. When transforming back to the usual units the question is whether effects are captured correctly. \

Now we analyze the pairwise correlation structure between the covariates as well as the covariates and the dependent variable.

```
pairs(train)
```



These correlations impact model selection as we can get a first impression of whether or not a linear model

would be a good assumption through the scatter plot. Here we can clearly see that `sex` is a categorical variable. We observe a clear linear relationship between `tc` and `ldl`. Therefore we might ask ourselves if these two variables are really independent predictors. Adding only one to the regression instead of both comes with a slight omitted variable bias, but can make sense for dependent variables in terms of variance reduction. In general, however, a linear relationship is not clearly observable. `tch` seems to be discrete. \

Next we fit a linear regression model containing all explanatory variables and inspect the model and evaluate the in sample fit as well as the performance on the test data based on the mean squared error (MSE).

```
model.full <- lm(y ~ ., data = train)
(summary.full <- summary(model.full))
```

```
##
## Call:
## lm(formula = y ~ ., data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -154.436  -37.748   -1.375   37.421  153.466
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  152.706      2.711  56.319  < 2e-16 ***
## age            9.856     62.721   0.157 0.875213
## sex         -240.347     64.936  -3.701 0.000245 ***
## bmi          499.266     70.415   7.090 6.35e-12 ***
## map          354.976     70.187   5.058 6.55e-07 ***
## tc          -861.163    436.264  -1.974 0.049095 *
## ldl          541.190    354.923   1.525 0.128119
## hdl          116.045    221.425   0.524 0.600518
## tch          166.516    166.601   0.999 0.318178
## ltg          773.896    179.728   4.306 2.11e-05 ***
## glu           63.631     68.817   0.925 0.355729
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 54.18 on 389 degrees of freedom
## Multiple R-squared:  0.5258, Adjusted R-squared:  0.5136
## F-statistic: 43.13 on 10 and 389 DF,  p-value: < 2.2e-16
```

```
(MSE.insample.full <- mean(summary.full$residuals^2))
```

```
## [1] 2854.869
```

```
outofsample.full <- data.frame(pred = predict(model.full,test[,-1]),
                    actual = test[,1])
(MSE.outofsample.full <- mean((outofsample.full$actual -
                        outofsample.full$pred)^2))
```

```
## [1] 2945.384
```

We observe that the out of sample fit is worse than the in sample fit.

Now we fit a smaller model where only the covariates are contained which according to a t-test are significant at the 5% significance level conditional on all other variables being included. We evaluate the performance in-sample as well as on the test data and compare this model to the full model using an F-test.

```
model.small <- lm(y ~ sex + bmi + map + tc + ltg, data = train)
(summary.small <- summary(model.small))
```

```
##
## Call:
## lm(formula = y ~ sex + bmi + map + tc + ltg, data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -154.487  -39.583   -2.167   36.677  143.460
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  152.676      2.744  55.634  < 2e-16 ***
## sex         -143.624     60.008  -2.393  0.01716 *
## bmi          580.467     67.332   8.621  < 2e-16 ***
## map          344.751     68.041   5.067 6.23e-07 ***
## tc          -218.311     67.313  -3.243  0.00128 **
## ltg          657.293     75.344   8.724  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 54.85 on 394 degrees of freedom
## Multiple R-squared:  0.5077, Adjusted R-squared:  0.5014
## F-statistic: 81.26 on 5 and 394 DF,  p-value: < 2.2e-16
```

```
(MSE.insample.small <- mean(summary.small$residuals^2))
```

```
## [1] 2963.644
```

```
outofsample.small <- data.frame(pred = predict(model.small,test[,-1]),
                      actual = test[,1])
(MSE.outofsample.small <- mean((outofsample.small$actual -
                          outofsample.small$pred)^2))
```

```
## [1] 3022.301
```

```
(F.small <- anova(model.full,model.small))
```

```
## Analysis of Variance Table
##
## Model 1: y ~ age + sex + bmi + map + tc + ldl + hdl + tch + ltg + glu
## Model 2: y ~ sex + bmi + map + tc + ltg
##   Res.Df     RSS Df Sum of Sq      F  Pr(>F)
## 1    389 1141947
## 2    394 1185458 -5    -43510 2.9643 0.01221 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Again we observe a higher out of sample MSE. The F-test suggests that the model improves when dropping insignificant coefficients. \

We now use stepwise regression based on the AIC to select a suitable model. The step function checks whether the AIC decreases when dropping variables in a stepwise procedure and stops as soon as it does not decrease anymore. We evaluate the performance in-sample as well as on the test data and compare this model to the full model using an F-test.

```
(model.stepwise <- step(model.full, criteria = "AIC"))
```

```
## Start:  AIC=3204.71
## y ~ age + sex + bmi + map + tc + ldl + hdl + tch + ltg + glu
##
##        Df Sum of Sq     RSS    AIC
## - age   1        72 1142020 3202.7
## - hdl   1       806 1142754 3203.0
## - glu   1      2510 1144457 3203.6
## - tch   1      2933 1144880 3203.7
## <none>              1141947 3204.7
## - ldl   1      6825 1148773 3205.1
## - tc    1     11438 1153386 3206.7
## - sex   1     40216 1182164 3216.6
## - ltg   1     54429 1196377 3221.3
## - map   1     75090 1217038 3228.2
## - bmi   1    147581 1289529 3251.3
##
## Step:  AIC=3202.74
## y ~ sex + bmi + map + tc + ldl + hdl + tch + ltg + glu
##
##        Df Sum of Sq     RSS    AIC
## - hdl   1       824 1142844 3201.0
## - glu   1      2656 1144676 3201.7
## - tch   1      2916 1144936 3201.8
## <none>              1142020 3202.7
## - ldl   1      6890 1148910 3203.1
## - tc    1     11478 1153497 3204.7
## - sex   1     40274 1182294 3214.6
## - ltg   1     54900 1196920 3219.5
## - map   1     79224 1221244 3227.6
## - bmi   1    147570 1289590 3249.3
##
## Step:  AIC=3201.03
## y ~ sex + bmi + map + tc + ldl + tch + ltg + glu
##
##        Df Sum of Sq     RSS    AIC
## - tch   1      2185 1145029 3199.8
## - glu   1      2705 1145549 3200.0
## <none>              1142844 3201.0
## - ldl   1      8808 1151653 3202.1
## - tc    1     27555 1170400 3208.6
## - sex   1     40811 1183656 3213.1
## - map   1     78720 1221564 3225.7
## - ltg   1     92523 1235368 3230.2
## - bmi   1    147071 1289915 3247.4
```

```
##
## Step:  AIC=3199.79
## y ~ sex + bmi + map + tc + ldl + ltg + glu
##
##        Df Sum of Sq     RSS    AIC
## - glu   1      3071 1148100 3198.9
## <none>              1145029 3199.8
## - ldl   1     36551 1181580 3210.4
## - sex   1     39159 1184188 3211.2
## - tc    1     61374 1206403 3218.7
## - map   1     76944 1221973 3223.8
## - bmi   1    146794 1291823 3246.0
## - ltg   1    239636 1384665 3273.8
##
## Step:  AIC=3198.86
## y ~ sex + bmi + map + tc + ldl + ltg
##
##        Df Sum of Sq     RSS    AIC
## <none>              1148100 3198.9
## - sex   1     37042 1185142 3209.6
## - ldl   1     37358 1185458 3209.7
## - tc    1     61253 1209352 3217.7
## - map   1     84790 1232890 3225.4
## - bmi   1    158343 1306443 3248.5
## - ltg   1    262231 1410331 3279.1
```

```
##
## Call:
## lm(formula = y ~ sex + bmi + map + tc + ldl + ltg, data = train)
##
## Coefficients:
## (Intercept)          sex          bmi          map           tc          ldl
##       152.7       -225.9        509.7        362.2       -775.9        554.5
##         ltg
##       805.3
```

```
(summary.stepwise <- summary(model.stepwise))
```

```
##
## Call:
## lm(formula = y ~ sex + bmi + map + tc + ldl + ltg, data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -157.214  -38.027   -2.143   36.163  149.530
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  152.723      2.704  56.477  < 2e-16 ***
## sex         -225.947     63.453  -3.561 0.000415 ***
## bmi          509.713     69.234   7.362 1.07e-12 ***
## map          362.152     67.222   5.387 1.23e-07 ***
## tc          -775.933    169.455  -4.579 6.28e-06 ***
```

```
## ldl          554.531    155.071   3.576 0.000392 ***
## ltg          805.250     84.993   9.474  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 54.05 on 393 degrees of freedom
## Multiple R-squared:  0.5232, Adjusted R-squared:  0.5159
## F-statistic: 71.88 on 6 and 393 DF,  p-value: < 2.2e-16
```

```
(MSE.insample.stepwise <- mean(summary.stepwise$residuals^2))
```

```
## [1] 2870.25
```

```
outofsample.stepwise <- data.frame(pred = predict(model.stepwise,test[,-1]),
                        actual = test[,1])
(MSE.outofsample.stepwise <- mean((outofsample.stepwise$actual -
                            outofsample.stepwise$pred)^2))
```

```
## [1] 2966.798
```

```
(F.stepwise <- anova(model.full,model.stepwise))
```

```
## Analysis of Variance Table
##
## Model 1: y ~ age + sex + bmi + map + tc + ldl + hdl + tch + ltg + glu
## Model 2: y ~ sex + bmi + map + tc + ldl + ltg
##   Res.Df     RSS Df Sum of Sq      F Pr(>F)
## 1    389 1141947
## 2    393 1148100 -4   -6152.4 0.5239 0.7182
```

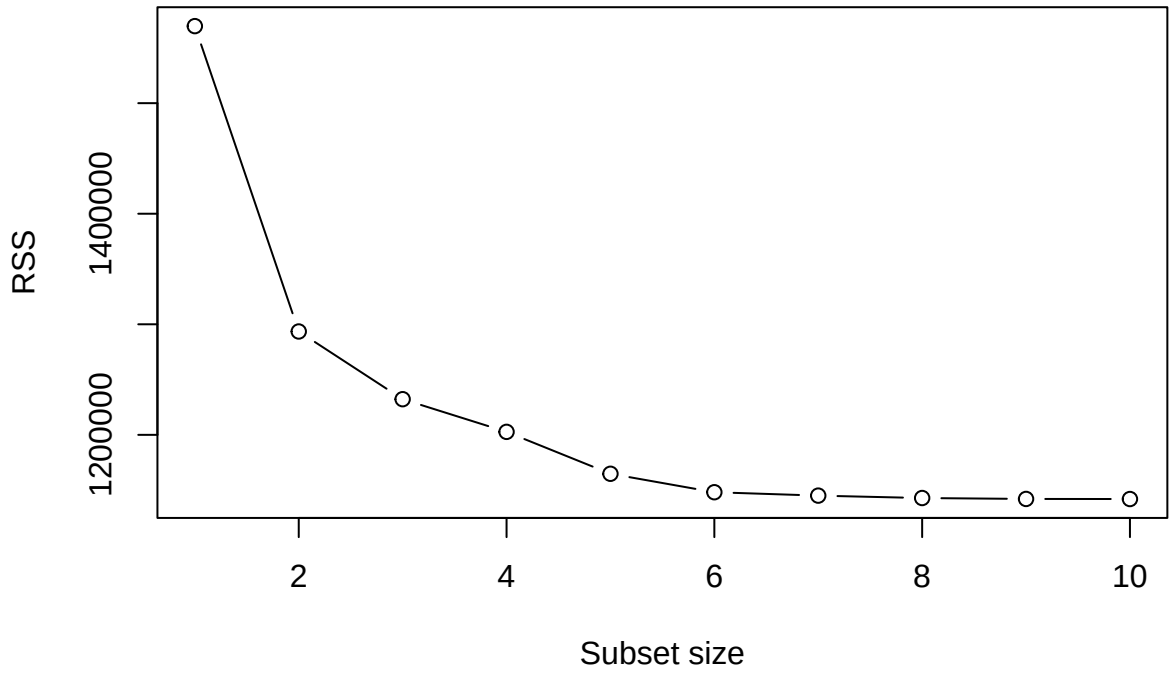
Lastly we use best subset selection to select a suitable model based on the AIC. We evaluate the performance in-sample as well as on the test data and compare this model to the full model using an F-test.

```
library(leaps)
subset <- regsubsets(y ~ .,train, nvmax = 10)
(subset.summary <- summary(subset))
```

```
## Subset selection object
## Call: regsubsets.formula(y ~ ., train, nvmax = 10)
## 10 Variables  (and intercept)
##     Forced in Forced out
## age     FALSE      FALSE
## sex     FALSE      FALSE
## bmi     FALSE      FALSE
## map     FALSE      FALSE
## tc      FALSE      FALSE
## ldl     FALSE      FALSE
## hdl     FALSE      FALSE
## tch     FALSE      FALSE
## ltg     FALSE      FALSE
## glu     FALSE      FALSE
```

```
## 1 subsets of each size up to 10
## Selection Algorithm: exhaustive
##           age sex bmi map tc  ldl hdl tch ltg glu
## 1  ( 1 )  " " " " "*" " " " " " " " " " " " " " "
## 2  ( 1 )  " " " " "*" " " " " " " " " " " "*" " "
## 3  ( 1 )  " " " " "*" "*" " " " " " " " " "*" " "
## 4  ( 1 )  " " " " "*" "*" "*" " " " " " " "*" " "
## 5  ( 1 )  " " "*" "*" "*" " " " " "*" " " "*" " "
## 6  ( 1 )  " " "*" "*" "*" "*" "*" " " " " "*" " "
## 7  ( 1 )  " " "*" "*" "*" "*" "*" " " " " "*" "*"
## 8  ( 1 )  " " "*" "*" "*" "*" "*" " " "*" "*" "*"
## 9  ( 1 )  " " "*" "*" "*" "*" "*" "*" "*" "*" "*"
## 10  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"
```

```
plot(subset.summary$rss, xlab = "Subset size", ylab = "RSS", type = "b")
```




```
data.frame(
  Adj.R2 = which.max(subset.summary$adjr2),
  CP = which.min(subset.summary$cp),
  BIC = which.min(subset.summary$bic)
)
```

```
##   Adj.R2 CP BIC
## 1      7  6   5
```

```r
#help function
get.model <- function(id, object, outcome){
  models <- summary(object)$which[id,-1]
  form <- as.formula(object$call[[2]])
  outcome <- all.vars(form)[1]
  predictors <- names(which(models == TRUE))
  predictors <- paste(predictors, collapse = "+")
  as.formula(paste0(outcome, "~", predictors))
}

#choose based on BIC
model.subset <- lm(get.model(5,subset,"y"), train)

(summary.subset <- summary(model.subset))
```

```
##
## Call:
## lm(formula = get.model(5, subset, "y"), data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -148.699  -38.009   -0.413   36.673  148.969
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   152.69       2.72  56.131  < 2e-16 ***
## sex          -233.35      63.90  -3.652 0.000295 ***
## bmi           506.03      68.90   7.344 1.20e-12 ***
## map           358.97      67.63   5.308 1.86e-07 ***
## hdl          -289.95      68.92  -4.207 3.21e-05 ***
## ltg           467.58      68.89   6.787 4.22e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 54.37 on 394 degrees of freedom
## Multiple R-squared:  0.5163, Adjusted R-squared:  0.5101
## F-statistic:  84.1 on 5 and 394 DF,  p-value: < 2.2e-16
```

```r
(MSE.insample.subset <- mean(summary.subset$residuals^2))
```

```
## [1] 2911.967
```

```r
outofsample.subset <- data.frame(pred = predict(model.subset,test[,-1]),
                      actual = test[,1])
(MSE.outofsample.subset <- mean((outofsample.subset$actual -
                          outofsample.subset$pred)^2))
```

```
## [1] 2956.157
```

```r
(F.subset <- anova(model.full,model.subset))
```

```
## Analysis of Variance Table
##
## Model 1: y ~ age + sex + bmi + map + tc + ldl + hdl + tch + ltg + glu
## Model 2: y ~ sex + bmi + map + hdl + ltg
##   Res.Df     RSS Df Sum of Sq     F Pr(>F)
## 1    389 1141947
## 2    394 1164787 -5    -22839 1.556 0.1716
```

We summarize our results in the following table.

```
library(reshape2)
library(plyr)
results <- join_all(list(melt(data.frame(as.list(model.full$coefficients))),
                         melt(data.frame(as.list(model.small$coefficients))),
                         melt(data.frame(as.list(model.stepwise$coefficients))),
                         melt(data.frame(as.list(model.subset$coefficients)))),
                    by="variable", type = "left")
colnames(results) <- c("value","full","small","stepwise","subset")
results <- data.frame(lapply(results, as.character), stringsAsFactors = F)

MSE.insample <- c("MSE in sample", MSE.insample.full, MSE.insample.small,
                MSE.insample.stepwise, MSE.insample.subset)
MSE.outofsample <- c("MSE out of sample", MSE.outofsample.full, MSE.outofsample.small,

results <- (rbind(results,MSE.insample,MSE.outofsample))
results <- cbind(results[,1],data.frame(lapply(results[,-1],
            function(x) round(as.numeric(x),digits = 2))))
rownames(results) <- results[,1]
(results <- results[,-1])
```

```
##                       full   small stepwise  subset
## X.Intercept.        152.71  152.68   152.72  152.69
## age                   9.86      NA       NA      NA
## sex                -240.35 -143.62  -225.95 -233.36
## bmi                 499.27  580.47   509.71  506.03
## map                 354.98  344.75   362.15  358.97
## tc                 -861.16 -218.31  -775.93      NA
## ldl                 541.19      NA   554.53      NA
## hdl                 116.05      NA       NA -289.96
## tch                 166.52      NA       NA      NA
## ltg                 773.90  657.29   805.25  467.58
## glu                  63.63      NA       NA      NA
## MSE in sample      2854.87 2963.64  2870.25 2911.97
## MSE out of sample 2945.38 3022.30  2966.80 2956.16
```

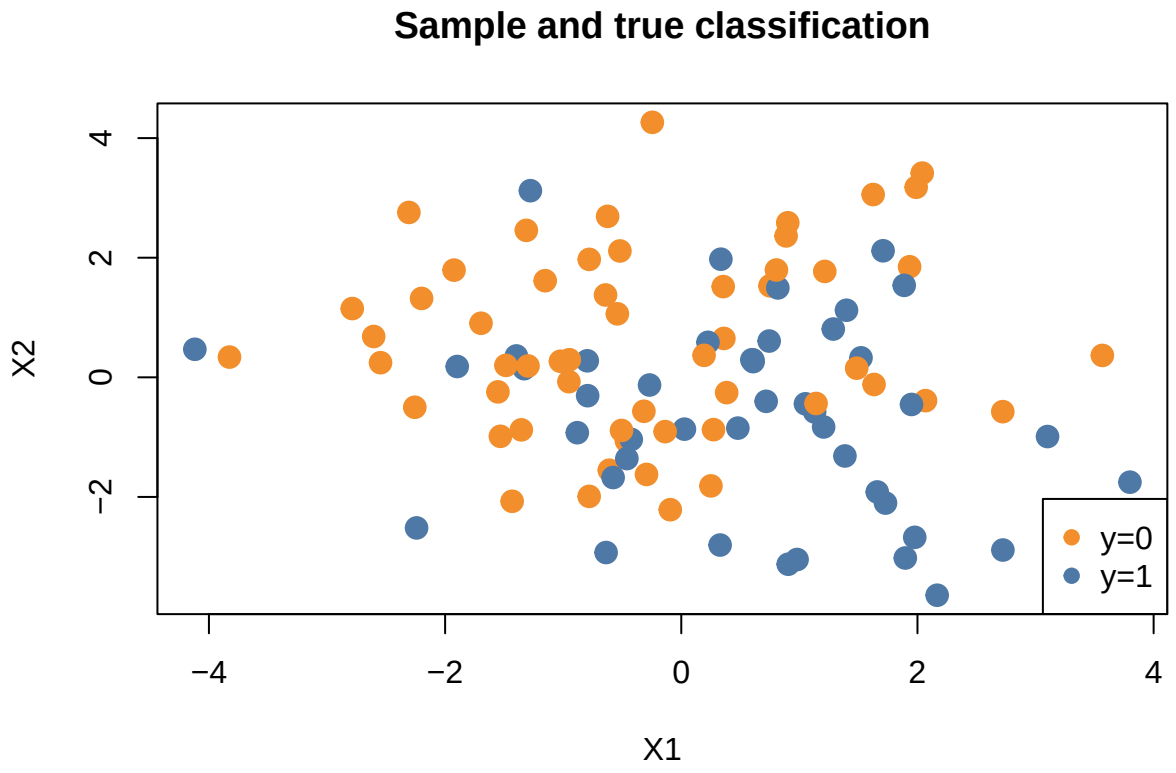
Exercise 4

We have the following data generating process

$$y \sim \text{Bernoulli}(0.5)$$
$$\mathbf{x} \sim \mathcal{N}\left(\begin{pmatrix} 0.5-y \\ -0.5+y \end{pmatrix}, \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}\right)$$

First we generate a sample of size 100 from this data generating process.

```
library(MASS)
sample <- function(n){
    y <- rbinom(n,1,0.5)
    mu <- lapply(y, function(x) c(0.5-x,-0.5+x))
    sigma <- matrix(c(2,0,0,2),nrow=2)
    x <- data.frame( matrix( unlist(lapply(mu, function(x)
         mvrnorm(1,x,sigma))), nrow = n, byrow = T ) )
    return(cbind(x,y))
}
data <-sample(100)

colors <- c("#4E79A7","#F28E2B")
with(data, plot(X1, X2, pch = 19,, cex = 1.5, col = colors[factor(y)]))
legend("bottomright",legend = paste0("y=",levels(factor(data$y))),
pch=19, col = colors[factor(data$y)])
title("Sample and true classification")
```



Then we determine the k-nearest neighbor prediction function given x for the class $y \in \{0,1\}$ for $k = 1,2,\ldots,100$. This means the prediction function results in either 0 or 1 depending on the class the majority of the observations have in the k-neighborhood.

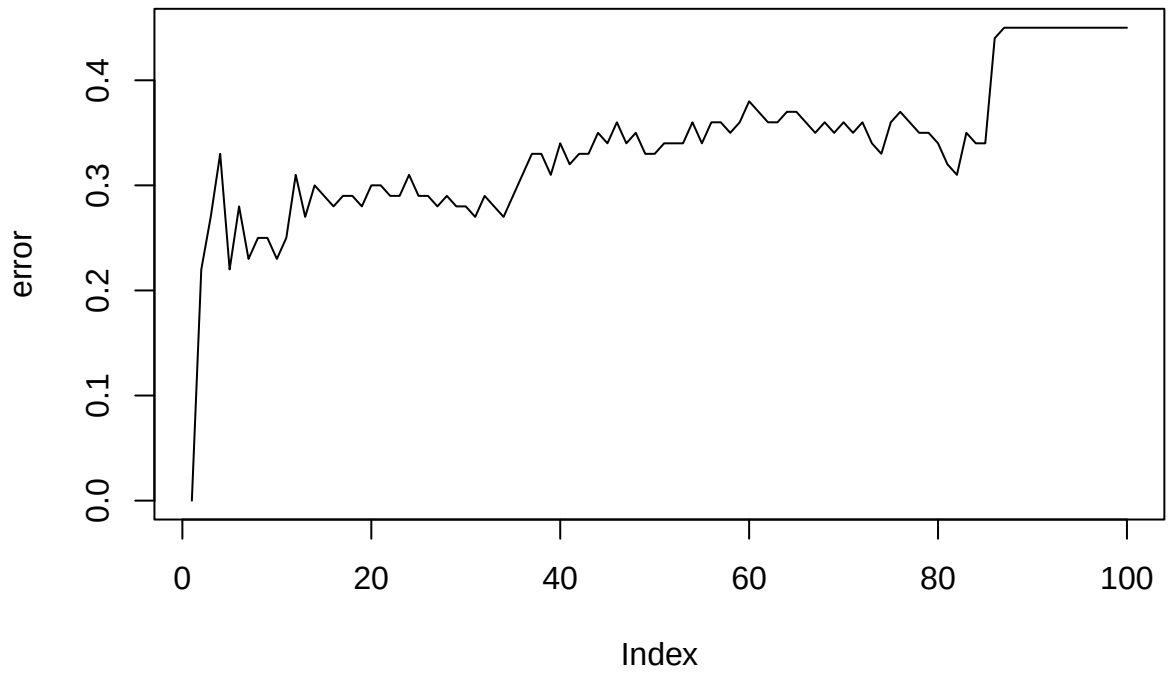
```
library(class)
mod <- lapply( 1:100, function(k) knn(data[,-3],data[,-3],cl=data[,3],k,prob=T) )
```

We compute the prediction error

$$\text{Error} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}(y_i \neq \hat{f}(\mathbf{x}_i))$$

for the observed values y and the predicted ones.

```
error <- unlist( lapply(mod, function(x) 1/length(x) * sum(x!=data[,3])) )
plot(error, type ="l")
```



Now we create a test data set through drawing a sample of size 1000 from the true data generating process.

```
test <- sample(1000)
mod.test <- lapply( 1:100, function(k) knn(data[,-3],test[,-3],cl=data[,3],k,prob=T) )
```

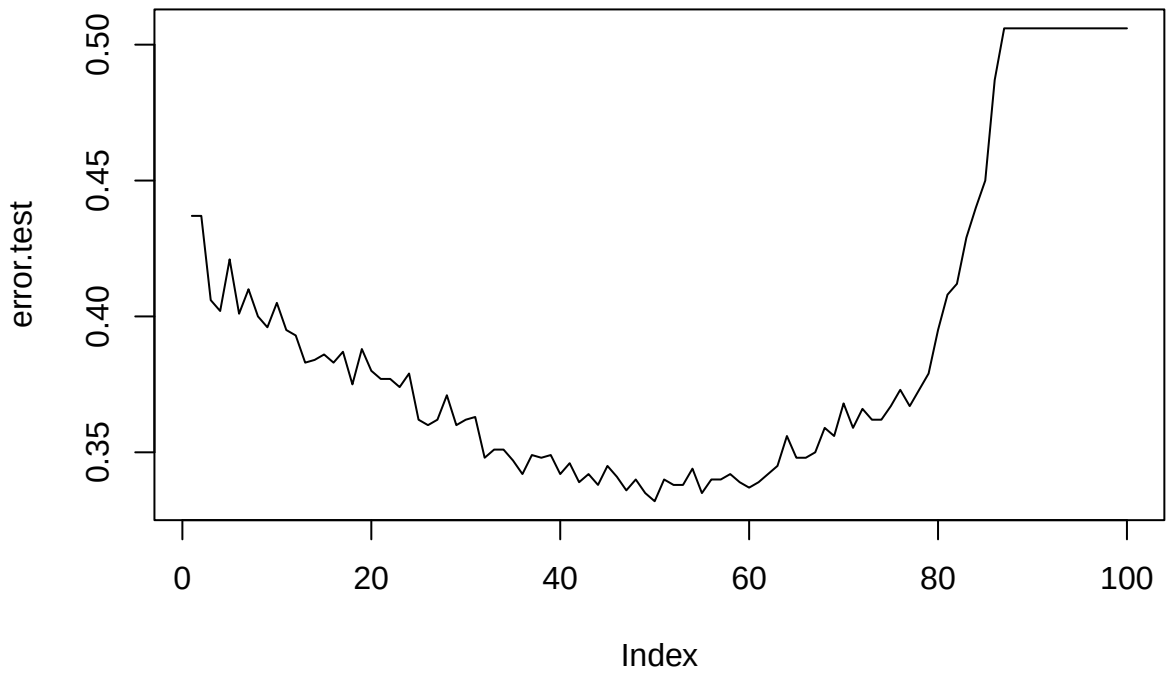
We calculate the error for the models on the test data set. We normalize the error values obtained by the estimates

$$\text{Error}_{\text{true}} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}(y_i \neq f(\mathbf{x}_i))$$

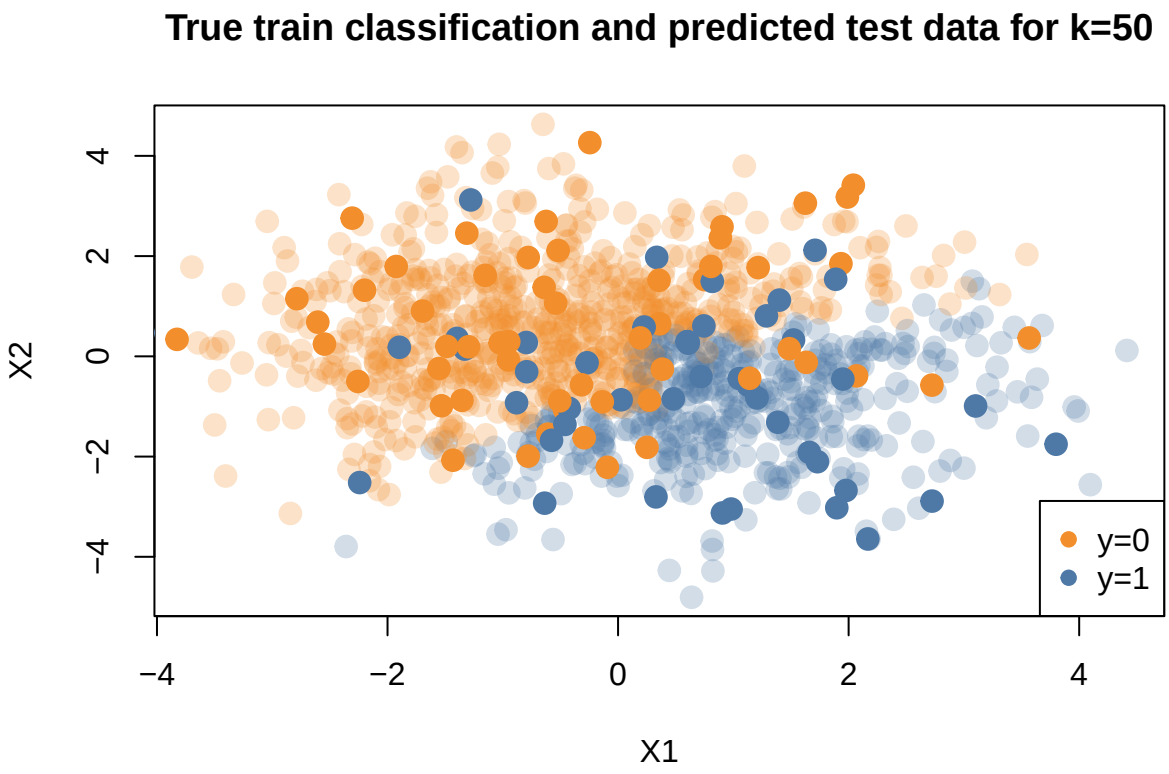
with

$$f(\mathbf{x}) = \mathbf{1}(x_2 \geq x_1).$$

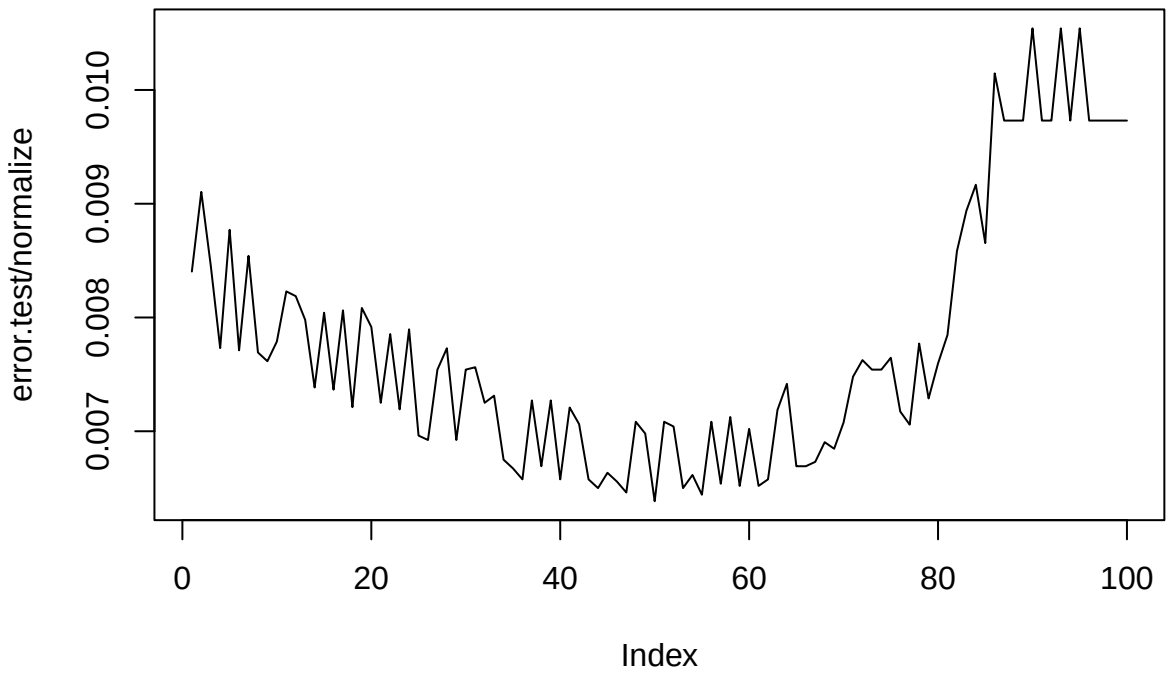
```
error.test <- unlist( lapply(mod.test, function(x) 1/length(x) * sum(x!=test[,3])) )
plot(error.test, type ="l")
```




```
colors <- c("#4E79A7","#F28E2B")
colors1 <- paste0(colors,"40")
y1 <- knn(data[,-3],test[,-3],cl=data[,3],50,prob=T)
data1 <- cbind(test[,-3],y1)
with(data1, plot(X1, X2, pch = 19, cex = 1.5, col = colors1[factor(y1)]))
with(data, points(X1,X2, pch = 19, cex = 1.5, col = colors[factor(y)]))
legend("bottomright",legend = paste0("y=",levels(factor(data$y))),
    pch=19, col = colors[factor(data$y)])
title("True train classification and predicted test data for k=50")
```

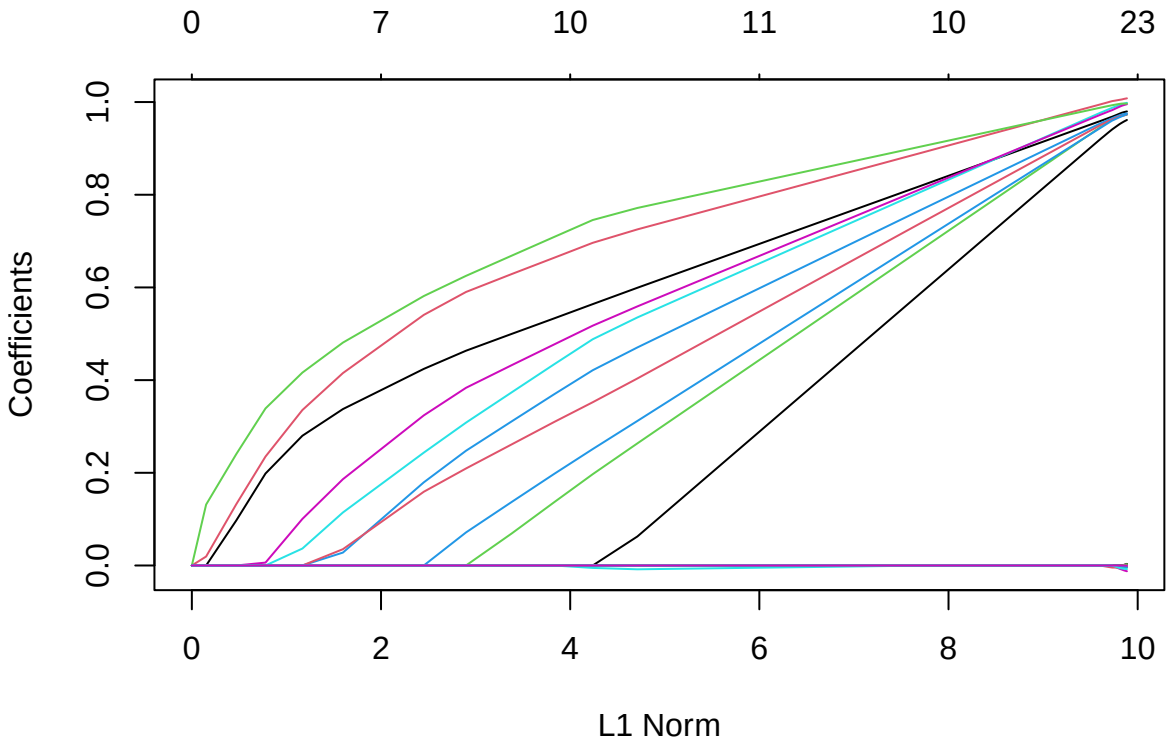



```
normalize <- unlist( lapply(data[,3], function(x) 1/length(x) *
sum(x!=as.numeric(data[,2] >= data[,1]) )) )
plot(error.test/normalize, type="l")
```



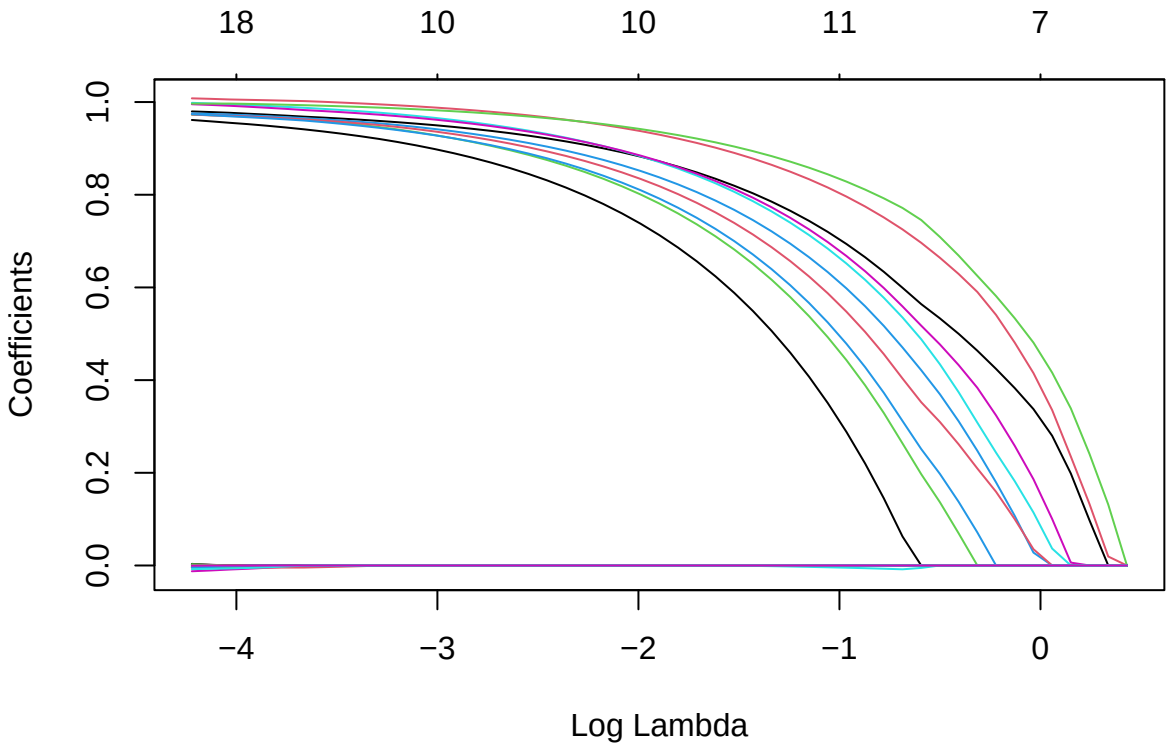
Exercise 5

```
library(MASS)
#set a random seed
set.seed(2412)
##sample size: 100 draws
n <- 100

#means of the variables = 0, for 100 variables (dimensions)
mean <- rep(0,100)

#covariance matrix - identity matrix 100x100
cov <- diag(100)

#our X matrix of covariates
X <- mvrnorm(n=n, mu=mean, Sigma = cov)

#draw 100 dependent variables
y <- rowSums(X[,1:10]) + rnorm(100, mean=0, sd=0.1)

library(glmnet)
lasso.fits <- glmnet(X, y)

plot(lasso.fits)
```



Each coloured curve represents a variable and its coefficients, depending on L1 Norm, the regularization term, which in turn depends on lambda, the complexity parameter. The smaller lambda, the less penalty, the closer the model to the OLS version (right-hand side of the figure). The larger lambda, the more penalty, the more coefficients shrink to zero. The axis above shows the number of non-zero coefficients for the respective lambda (i.e. the edf for the fit).

```
plot(lasso.fits, xvar = "lambda")
```



This shows the same as in the previous figure, just on another scale. \ \ We determine the number of non-zero coefficients in dependence of λ.

```
lasso.fits$lambda #from high lambda to low lambda
```

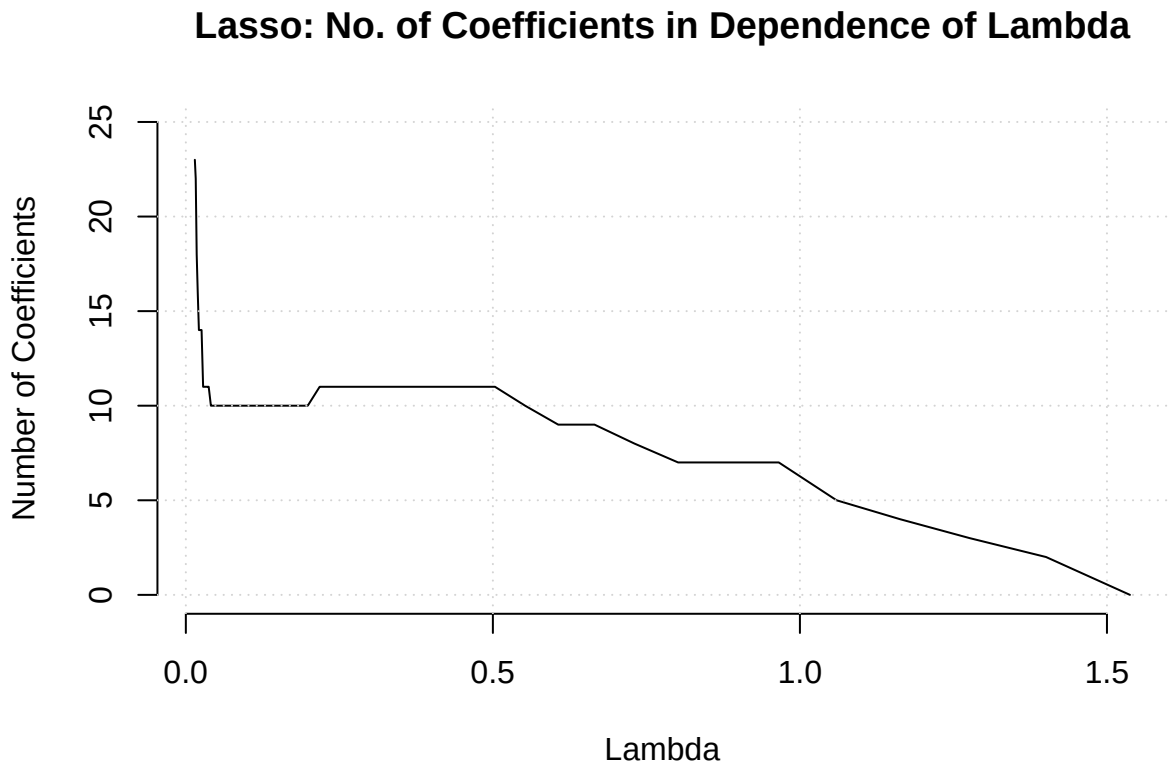
```
##  [1] 1.53744688 1.40086433 1.27641541 1.16302218 1.05970250 0.96556145
##  [7] 0.87978363 0.80162608 0.73041182 0.66552405 0.60640073 0.55252976
## [13] 0.50344454 0.45871991 0.41796850 0.38083733 0.34700479 0.31617784
## [19] 0.28808947 0.26249640 0.23917694 0.21792912 0.19856890 0.18092859
## [25] 0.16485539 0.15021009 0.13686584 0.12470706 0.11362843 0.10353399
## [31] 0.09433631 0.08595574 0.07831967 0.07136196 0.06502236 0.05924596
## [37] 0.05398271 0.04918703 0.04481739 0.04083594 0.03720819 0.03390271
## [43] 0.03089089 0.02814663 0.02564616 0.02336783 0.02129189 0.01940038
## [49] 0.01767690 0.01610654 0.01467568
```

```
selected <- predict(lasso.fits, type = "nonzero") #predicts model with all lambdas
#from high lambda=high penalisation=few variables to low lambda=close to OLS

#plot lambdas against number of selected variables i.e. for smallest lambda 23 variables
#selected etc.
lengths(selected)
```

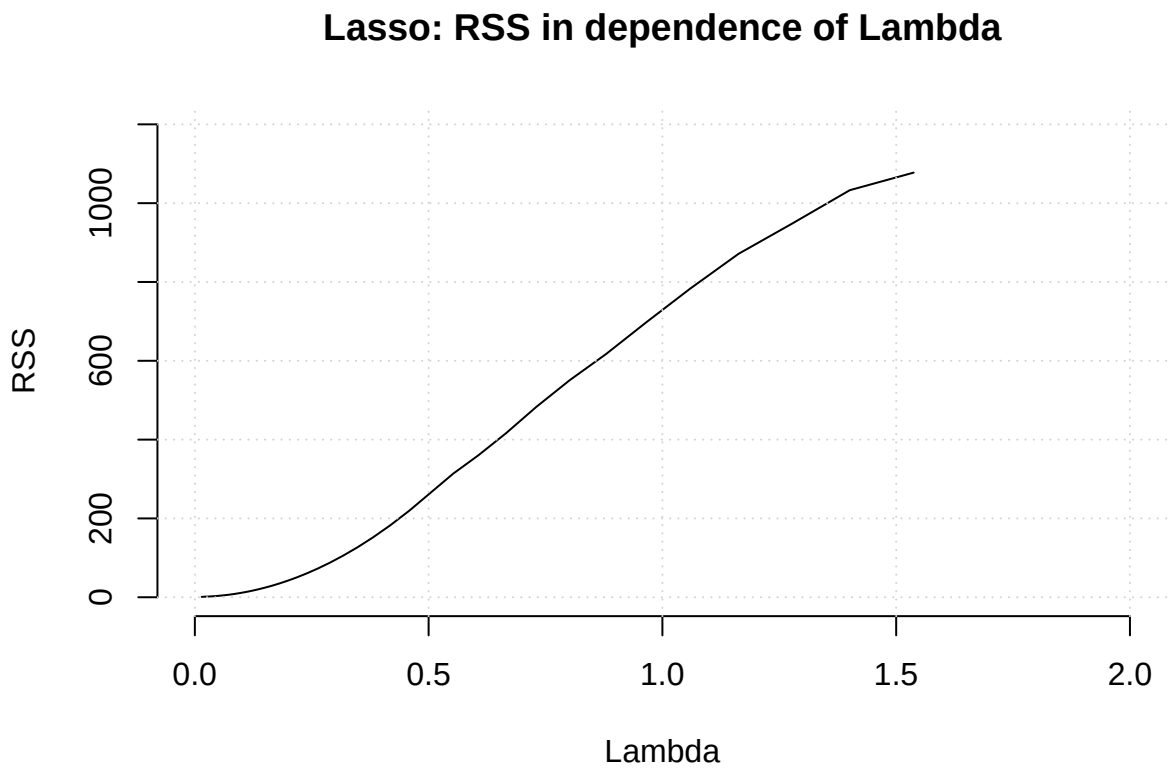
```
##  s0  s1  s2  s3  s4  s5  s6  s7  s8  s9 s10 s11 s12 s13 s14 s15 s16 s17 s18 s19
##   0   2   3   4   5   7   7   7   8   9   9  10  11  11  11  11  11  11  11  11
## s20 s21 s22 s23 s24 s25 s26 s27 s28 s29 s30 s31 s32 s33 s34 s35 s36 s37 s38 s39
```

```
##  11  11  10  10  10  10  10  10  10  10  10  10  10  10  10  10  10  10  10  10
## s40 s41 s42 s43 s44 s45 s46 s47 s48 s49 s50
##  11  11  11  11  14  14  14  16  18  22  23
```

```
plot(lasso.fits$lambda, lengths(selected), xlab = "Lambda",
     ylab = "Number of Coefficients", bty="n", ylim = c(0,25),
     main = "Lasso: No. of Coefficients in Dependence of Lambda", type="l")
grid()
```




```
rss <- deviance(lasso.fits) #rss for various lambdas

plot(lasso.fits$lambda, rss, xlab = "Lambda", ylab = "RSS", bty="n", xlim = c(0,2),
ylim = c(0,1200), main = "Lasso: RSS in dependence of Lambda", type="l")
grid()
```



Additionally, we wanted to know how many unique variables are selected along the different values of lambda.

```
length(lasso.fits$lambda) #we have 51 different lambdas
```

```
## [1] 51
```

```
length(unlist(selected))
```

```
## [1] 526
```

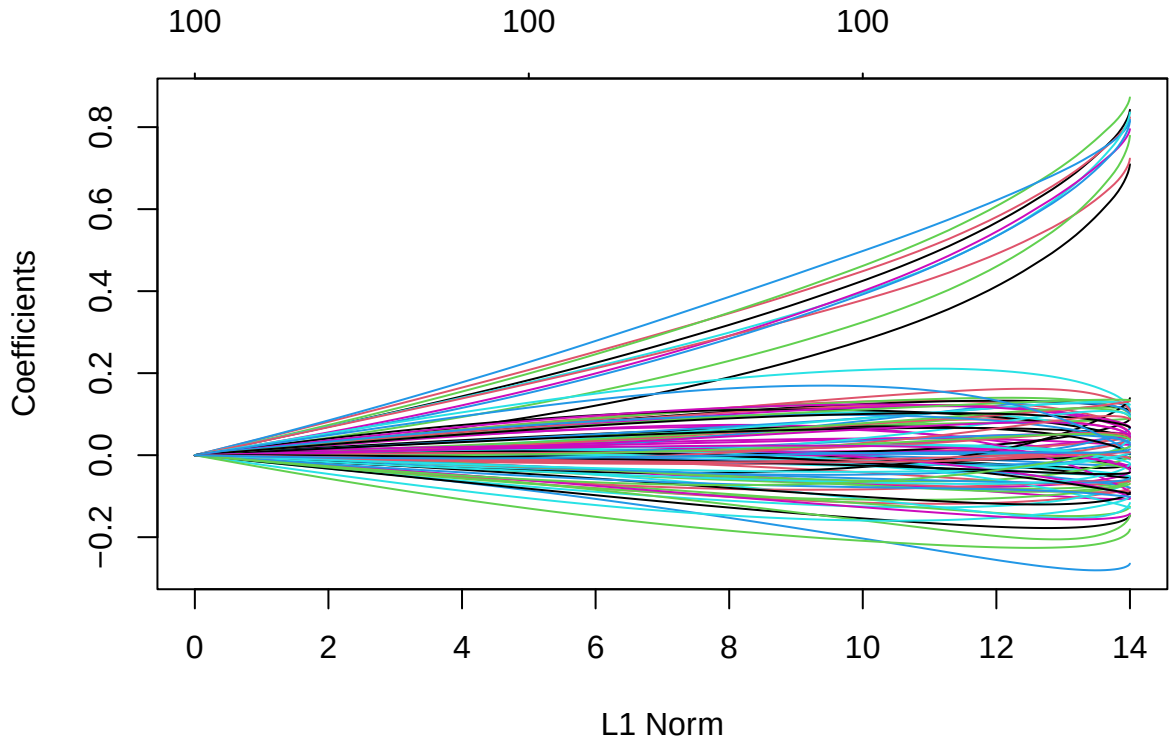
```
#in total, these lambdas lead to 526 non-zero coefficients
#however, the respective variables are counted multiple times
length(unique(unlist(selected)))
```

```
## [1] 24
```

```
#We have 24 unique variables (23 of them used in the version with the smallest lambda,
#see length(selected$s50))
```

Exercise 6

```
set.seed(101)
##sample size: 100 draws
n2 <- 100
#means of the variables = 0, for 100 variables (dimensions)
mean2 <- rep(0,100)
#covariance matrix - identity matrix 100x100
cov2 <- diag(100)
#our X matrix of covariates
X2 <- mvrnorm(n=n2, mu=mean2, Sigma = cov2)
#draw 100 dependent variables
y2 <- rowSums(X2[,1:10]) + rnorm(100, mean=0, sd=0.1)

ridge.fits <- glmnet(X2, y2, alpha=0)
plot(ridge.fits)
```



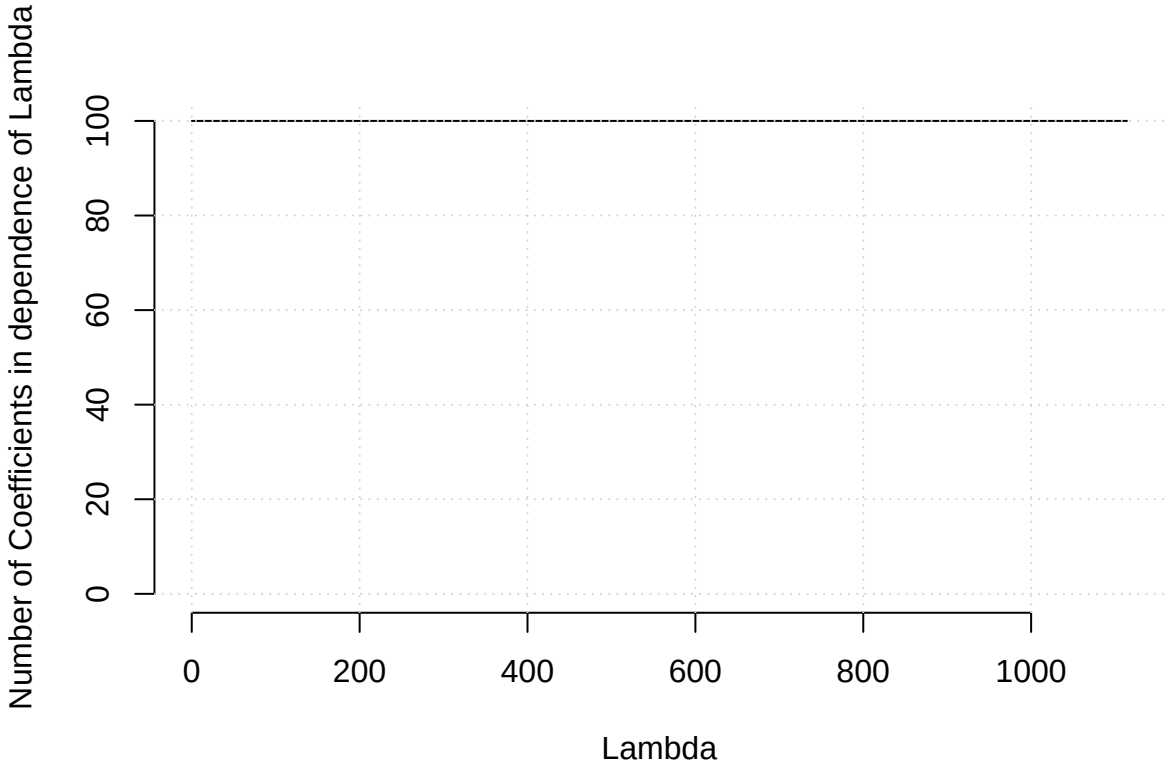
Each variable is represented by one line. The size of the respective coefficients varies with the penalisation term (L2-norm) depending on λ. The higher λ, the more penalisation, the lower are the coefficients. The lower λ, the higher the coefficients and the closer the solution to the OLS results. All variables are shrunken but variables with higher coefficients are shrunken relatively more. The coefficients converge to zero but do not get zero (see graph below where we have 100 variables for each value of lambda).

```
ridge.fits$lambda #from high lambda to low lambda
```

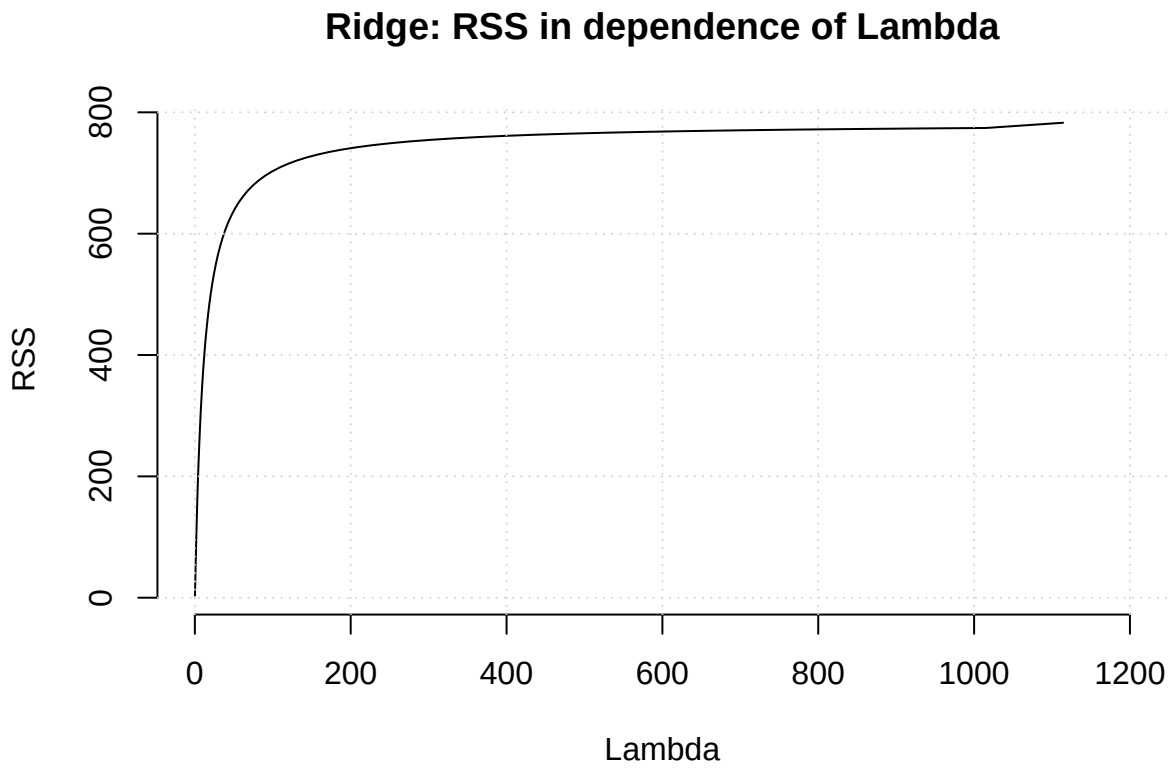
```
##   [1] 1114.1419737 1015.1646715  924.9802400  842.8075448  767.9348454
##   [6]  699.7136303  637.5529998  580.9145485  529.3077011  482.2854637
```

```
##  [11]  439.4405524  400.4018649  364.8312667  332.4206625  302.8893271
##  [16]  275.9814740  251.4640405  229.1246682  208.7698642  190.2233248
##  [21]  173.3244089  157.9267462  143.8969693  131.1135591  119.4657919
##  [26]  108.8527802   99.1825992   90.3714905   82.3431363   75.0279991
##  [31]   68.3627184   62.2895629   56.7559298   51.7138894   47.1197700
##  [36]   42.9337795   39.1196609   35.6443780   32.4778297   29.5925888
##  [41]   26.9636648   24.5682871   22.3857082   20.3970236   18.5850082
##  [46]   16.9339673   15.4296004   14.0588772   12.8099253   11.6719268
##  [51]   10.6350250    9.6902387    8.8293846    8.0450064    7.3303102
##  [56]    6.6791057    6.0857523    5.5451109    5.0524985    4.6036485
##  [61]    4.1946730    3.8220298    3.4824912    3.1731163    2.8912254
##  [66]    2.6343769    2.4003461    2.1871060    1.9928095    1.8157738
##  [71]    1.6544655    1.5074873    1.3735663    1.2515425    1.1403589
##  [76]    1.0390525    0.9467460    0.8626397    0.7860051    0.7161786
##  [81]    0.6525553    0.5945841    0.5417629    0.4936341    0.4497810
##  [86]    0.4098237    0.3734161    0.3402429    0.3100166    0.2824756
##  [91]    0.2573812    0.2345162    0.2136824    0.1946995    0.1774029
##  [96]    0.1616429    0.1472830    0.1341988    0.1222769    0.1114142
```

```r
selected2 <- predict(ridge.fits, type = "nonzero")

plot(ridge.fits$lambda, lengths(selected2), xlab = "Lambda",
     ylab = "Number of Coefficients in dependence of Lambda",
     bty="n", ylim = c(0,100), type="l")
grid()
```



```
rss2 <- deviance(ridge.fits)
plot(ridge.fits$lambda, rss2, xlab = "Lambda", ylab = "RSS", bty="n", xlim = c(0,1200),
main = "Ridge: RSS in dependence of Lambda", type="l")
grid()
```




```
length(unique(unlist(selected2)))
```

```
## [1] 100
```

```
#100 unique values
```