Gruppe 37

Emil Myhre og Lisa Erfjord

19 2 2019

Problem 1: Multiple linear regression

Model output

```
library(GLMsData)
data("lungcap")
lungcap$Htcm=lungcap$Ht*2.54
modelA = lm(log(FEV) ~ Age + Htcm + Gender + Smoke, data=lungcap)
modelB = lm(FEV ~ Age + Htcm + Gender + Smoke, data=lungcap)
#summary(modelA)
#summary(modelB)
```

Q1: The equation for the fitted modelA is given by

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \epsilon,$$

where the response Y = $log(FEV)$ and covariates x_1 = Age, x_2 = Htcm, x_3 = Gender and x_4 = Smoke. ϵ is an error term which is caused by all unconsidered covariates.

From the summary we have are presented the fitted values

$$Y = \ -1.943998 + 0.023387 x_1 + 0.016849 x_2 + 0.029319 x_3 - 0.046067 x_4$$

Q2:

ESTIMATE: the predicted values of the regression coefficients, given by $\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$. Meaning that if a covariate is increased by 1, the response is increased with the value of the corresponding regression coefficient. The intercept is the expected value of the response if no covariates are considered, that is, they are set to be zero. In this case, it would not make sense if for instance age or Htcm was equal to zero.

STD.ERROR: The standard error, or the standard deviation of the coefficients, is the squareroot of the variance. That is, the squareroot of the diagonal entries of the covariance matrix given by $\hat{\sigma}^2(\mathbf{X}^T\mathbf{X})^{-1}$, where $\hat{\sigma} = \text{RSS}/(n-p-1)$.

RESIDUAL STD.ERROR: The estimated value of the standard deviation of the error term ϵ. The residual standard error is given by RSS/$(n-p-1)$ where RSS=$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$.

F-STATISTIC: Given by

$$F = \frac{(\text{TSS-RSS})/p}{\text{RSS}/(n-p-1)} \sim F_{p,n-p-1}$$

It is used for hypothesis testing in linear regression, where we are interested in the null hypothesis that all regression coefficients are zero, $H_0 : \beta_j = 0$ for all j. We can calculate the p-value, denoted p, by $\text{p} = P(F_{p,n-p-1} > \tilde{f})$, where $\tilde{f}$ is the numerical value of F with our observed data. If the p-value is sufficiently low, that is it is smaller than the chosen significance level, we can reject the null hypothesis and conclude that at least one of the covariates affects the response.

Model fit

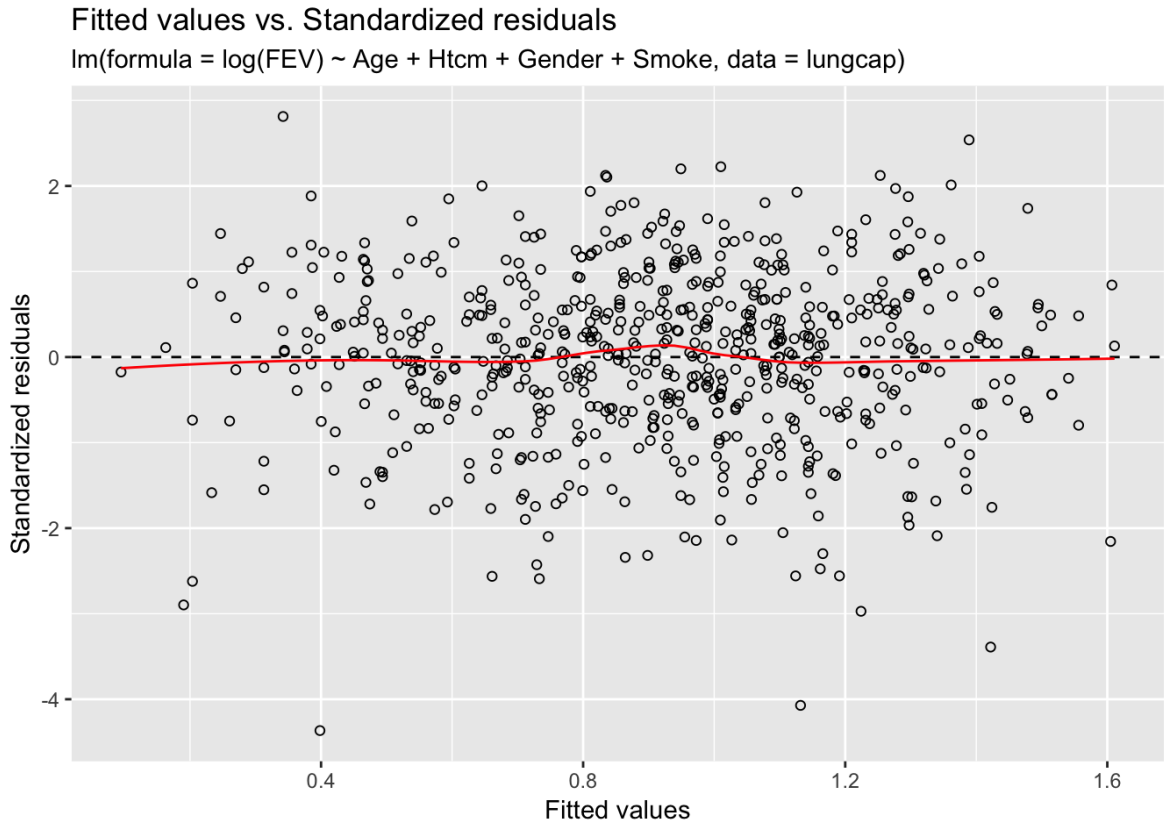
Q3: The fraction of variablity explained by the fitted model is given by the R^2 statistic defined as

$$R^2 = \frac{\text{TSS} - \text{RSS}}{TSS} = 1 - \frac{\text{RSS}}{\text{TSS}} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}.$$

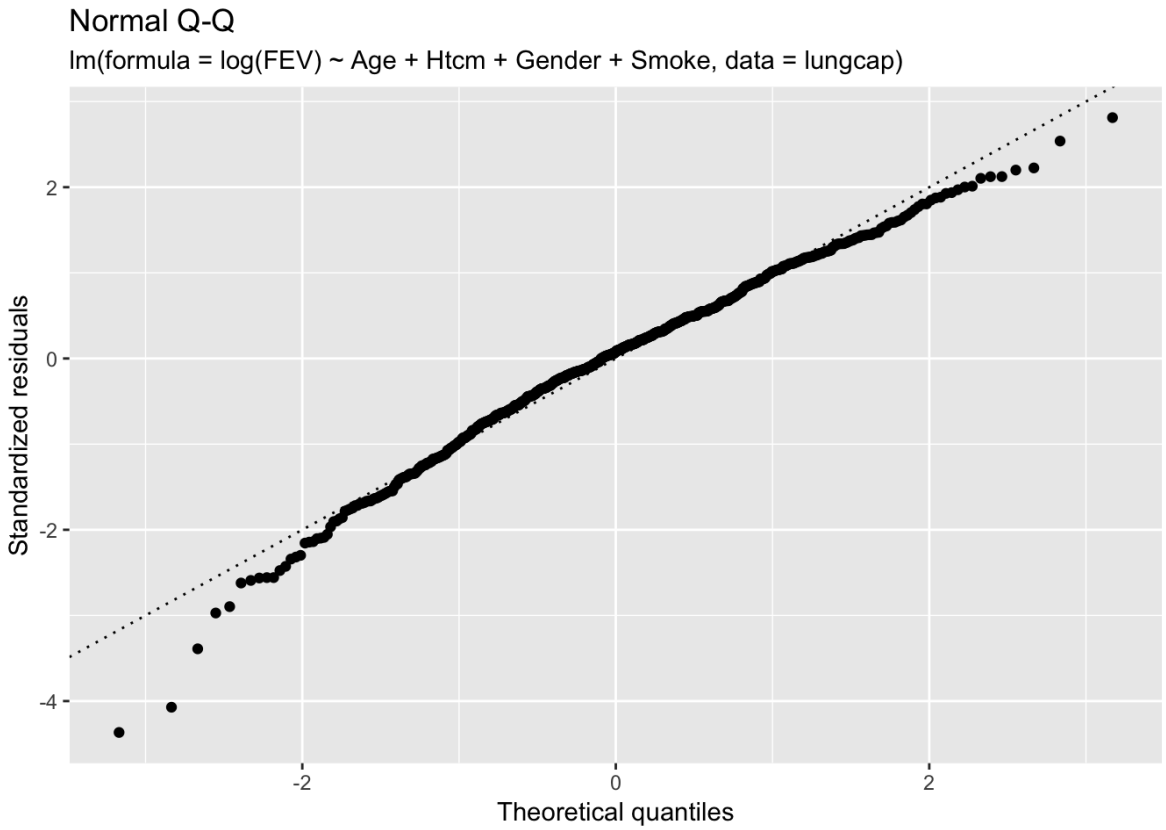
R^2 ranges from 0 to 1, indicating the portion of variability explained. So if $R^2 = 1$ all variation is explained by the model. For model A we see that $R^2 = 0.81$, which is pretty good and indicates a good model.

Q4:

```
library(ggplot2)
# residuls vs fitted
ggplot(modelA, aes(.fitted, .stdresid)) + geom_point(pch = 21) +
  geom_hline(yintercept = 0, linetype = "dashed") +
  geom_smooth(se = FALSE, col = "red", size = 0.5, method = "loess") +
  labs(x = "Fitted values", y = "Standardized residuals",
       title = "Fitted values vs. Standardized residuals",
       subtitle = deparse(modelA$call))
```




```
# qq-plot of residuals
ggplot(modelA, aes(sample = .stdresid)) +
  stat_qq(pch = 19) +
  geom_abline(intercept = 0, slope = 1, linetype = "dotted") +
  labs(x = "Theoretical quantiles", y = "Standardized residuals",
       title = "Normal Q-Q", subtitle = deparse(modelA$call))
```




```
# normality test
library(nortest)
ad.test(rstudent(modelA))
```

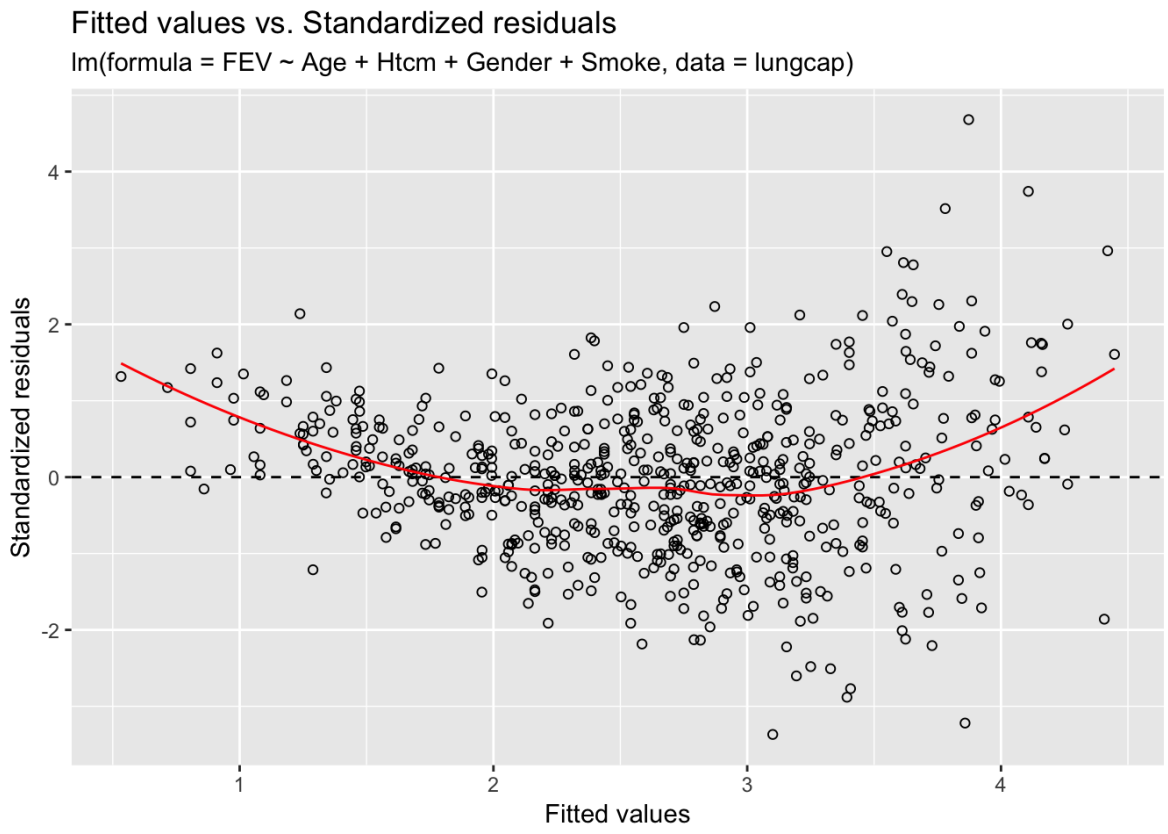
```
##
##  Anderson-Darling normality test
##
## data:  rstudent(modelA)
## A = 1.9256, p-value = 6.486e-05
```

From the “fitted values vs standardized residuals” plot, we see that the residuals are pretty evenly spread around a linear line. This indicates a linear relationship between the covariates and the response, which is a good indicator to the model.

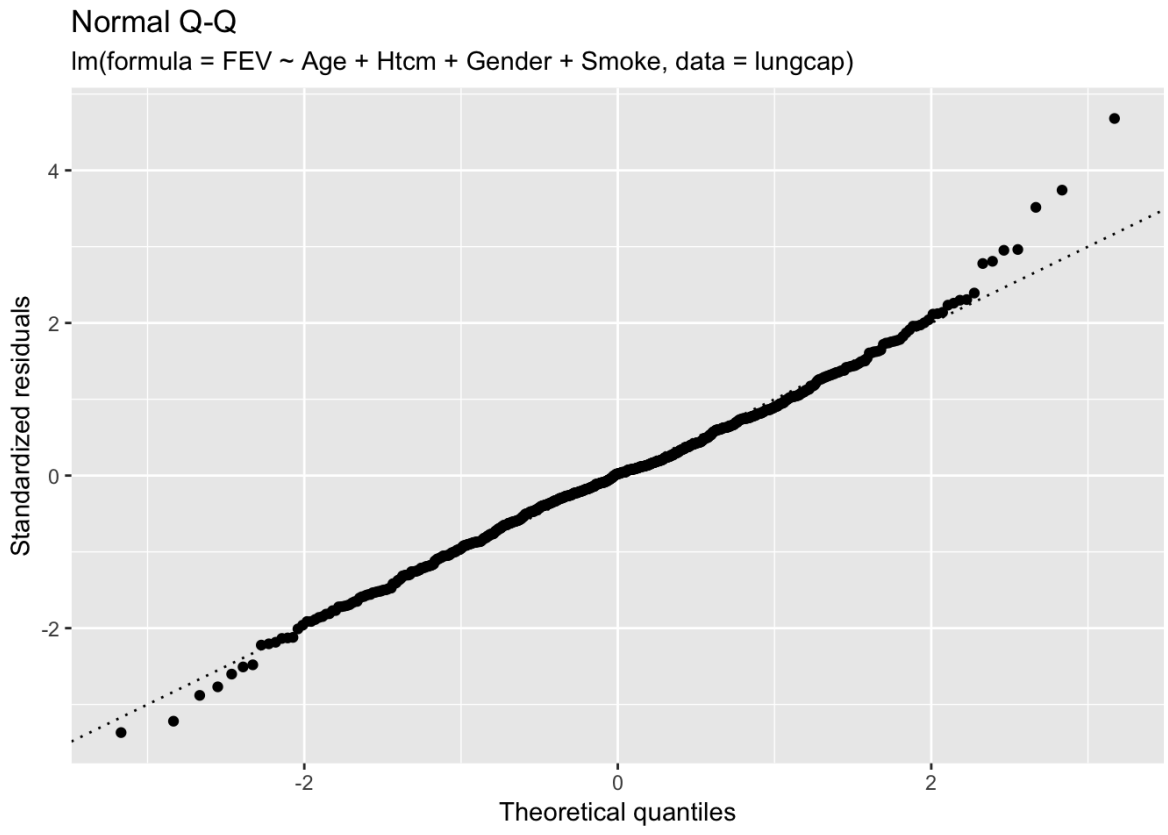
From the “normal Q-Q” plot, we see that the residuals deviates from the straight line at the tails. This indicates that the residuals are not normally distributed. This is also confirmed by the Anderson-Darling normality test, which gives a very low p-value.

Q5:

```
library(ggplot2)
# residuls vs fitted
ggplot(modelB, aes(.fitted, .stdresid)) + geom_point(pch = 21) +
  geom_hline(yintercept = 0, linetype = "dashed") +
  geom_smooth(se = FALSE, col = "red", size = 0.5, method = "loess") +
  labs(x = "Fitted values", y = "Standardized residuals",
       title = "Fitted values vs. Standardized residuals",
       subtitle = deparse(modelB$call))
```




```
# qq-plot of residuals
ggplot(modelB, aes(sample = .stdresid)) +
  stat_qq(pch = 19) +
  geom_abline(intercept = 0, slope = 1, linetype = "dotted") +
  labs(x = "Theoretical quantiles", y = "Standardized residuals",
       title = "Normal Q-Q", subtitle = deparse(modelB$call))
```



```
# normality test
 library(nortest)
ad.test(rstudent(modelB))
```

```
##
##  Anderson-Darling normality test
##
## data:  rstudent(modelB)
## A = 1.2037, p-value = 0.003853
```

We see from the “fitted values vs standardized residuals” of modelB, that the linear relationship between covariates and response does not seem to hold. In addition, from the “normal Q-Q” plot and the Anderson-Darling normality test which gives a p-value of $0.038 \leq 0.05$, we can conclude that the residuals are not normally distributed.

To evaluate which model we prefer it is possible to look at the R^2 value. Since the R^2 value explaines how much of the variance the model explaines, we prefer the model with the greatest value. For model A $R_A^2 = 0.81$ and for model B $R_B^2 = 0.775$, so model A is the preferred one. Another aspect one can look at is that model A has at least one linear relationship between the covariates and the response. This suggest that model A is a better model as well.

Hypothesis test and confidence interval

Q6:

```
summary(modelA)
```

We will perform a p-test for this. From the summary of modelA we see that the p-value of the covariate Age is $7.1 * 10^{-12}$. Due to this very low value, the null hypothesis can be rejected with significance level 0.001.

Q7:

```
confint(modelA,level=0.99)
```

```
##                    0.5 %       99.5 %
## (Intercept) -2.1471551289 -1.740841225
## Age          0.0147367391  0.032037689
## Htcm         0.0151410623  0.018556409
## GenderM     -0.0009546847  0.059593401
## Smoke       -0.1000874831  0.007952411
```

A 99% confidence interval for the covariate Age is given by

$$\hat{\beta}_{Age} \pm t_{0.005,\, n-2} \cdot \mathrm{SE}(\hat{\beta}_{Age})$$

Numerically we have the interval $[0.0147367391, 0.032037689]$. This means that with a 99% certainty the actual value for β_{Age} will be within this interval. Since the interval does not contain zero we can with 99% certainty conclude that $\beta_{Age} \neq 0$. Thus we can with 0.01 significance level reject the nullhypothosis, $H_0 : \beta_{Age} = 0$.

Prediction

Q8:

```
new = data.frame(Age=16, Htcm=170, Gender="M", Smoke=0)

f.ci = predict(modelA, newdata = new, level = 0.95, interval = "prediction")
f.ci[2] = exp(f.ci[2])
f.ci[3] = exp(f.ci[3])
f.ci
```

```
##        fit      lwr      upr
## 1 1.323802 2.818373 5.010038
```

` The best guess for his $log(FEV)$ is 1,323802. The 95% prediction interval for FEV is given by $[2.818373, 5.010038]$. This means that with 95% certainty the boy would have a FEV within the prediction interval. However, the range of the interval is so huge that it is not really too useful.

Problem 2: Classification

KNN classification

```
library(class)
library(caret)
```

```
## Loading required package: lattice
```

```
# for confusion matrices

raw = read.csv("https://www.math.ntnu.no/emner/TMA4268/2019v/data/tennis.csv")
M = na.omit(data.frame(y=as.factor(raw$Result),
                       x1=raw$ACE.1-raw$UFE.1-raw$DBF.1,
                       x2=raw$ACE.2-raw$UFE.2-raw$DBF.2))
set.seed(4268) # for reproducibility
tr = sample.int(nrow(M),nrow(M)/2)
trte=rep(1,nrow(M))
trte[tr]=0
Mdf=data.frame(M,"istest"=as.factor(trte))
```

Q9: For a positive integer K let $\mathcal{N}$ be the K points in the training data nearest to $x = (x_1, x_2)$. The point x is classified by taking a majority vote of the neighbors. That is, the KNN estimate the posterior class probability as

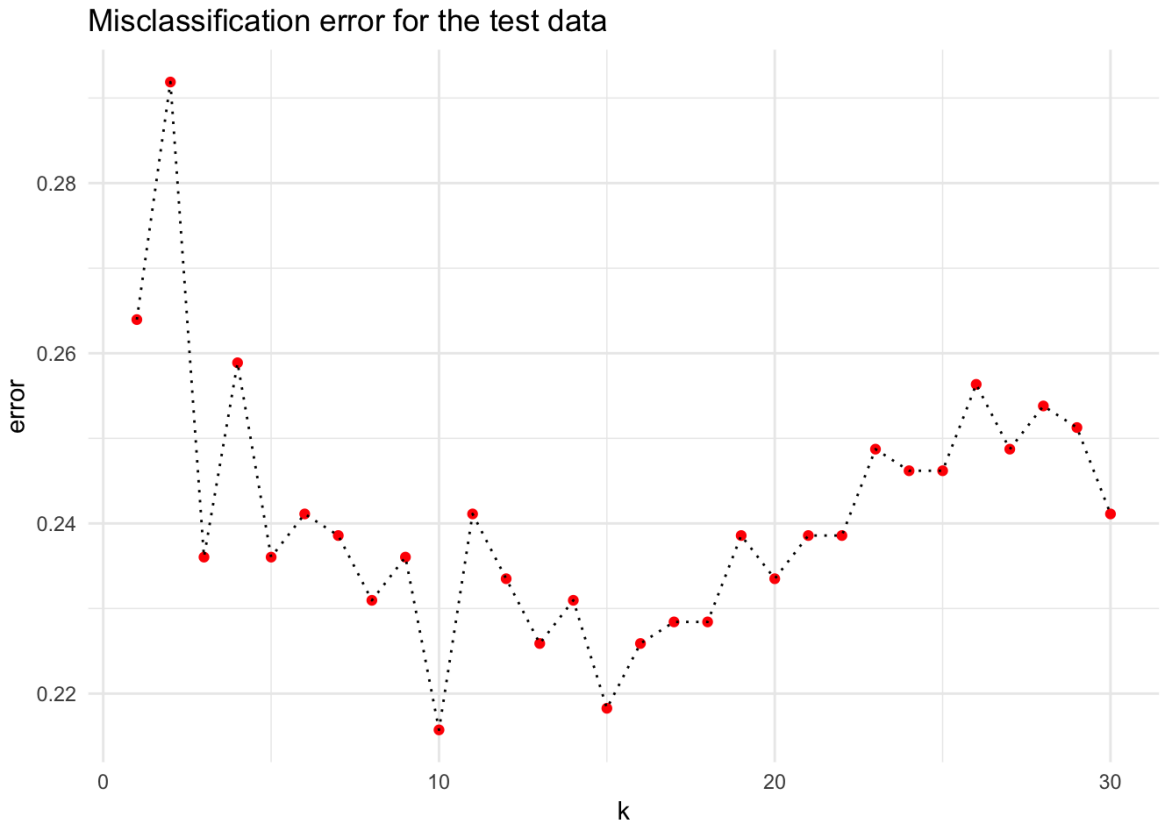
$$\hat{P}(Y = j \,|\, X = x) = \frac{1}{K} \sum_{i \in \mathcal{N}} I(y_i = j)$$

for $j = \{0, 1\}$, where I equals 1 if $y_i = j$ and 0 if $y_i \neq j$. So the KNN estimator $\hat{y}(x) \in \{0, 1\}$ is set to be the class with the largest probability from the formula above.

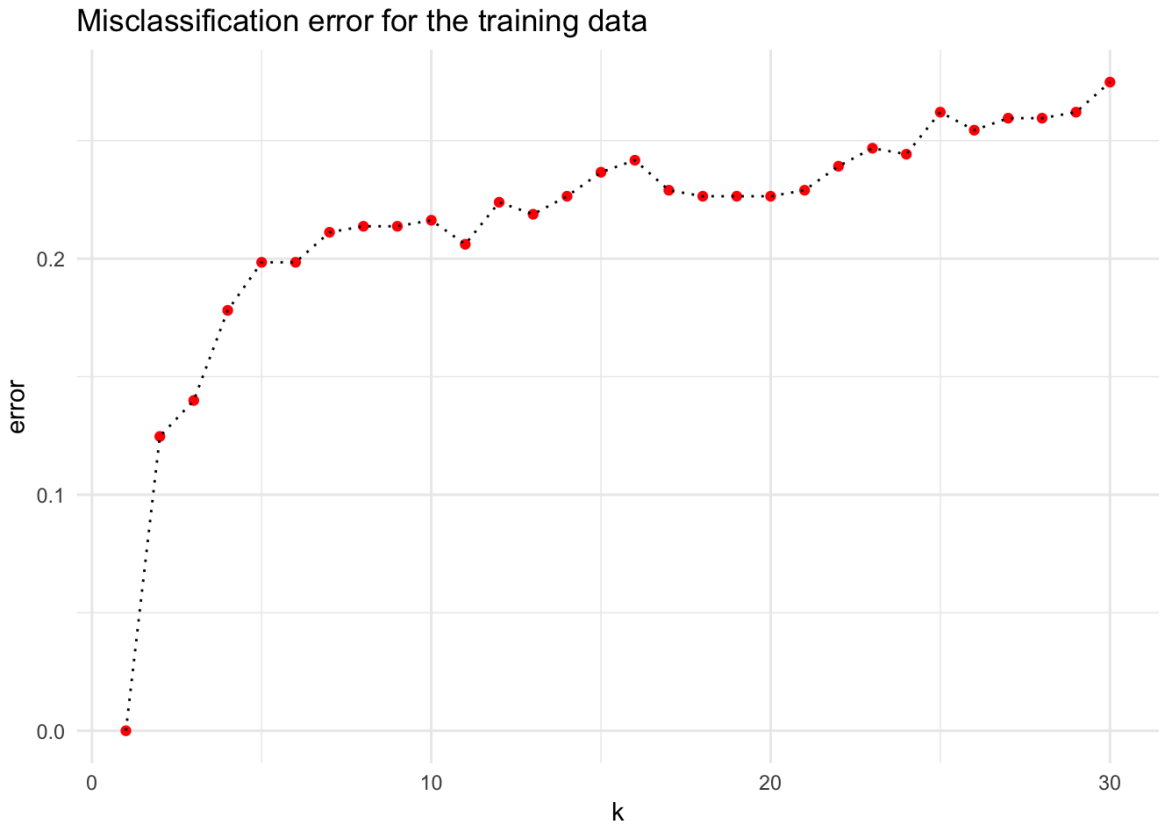
Q10: The misclassification error for the training data and the test data using KNN classification for all k, where $k \in \{1, 2, \ldots, 30\}$, is computed below.

```
# Deler opp dataene i training og test
trainingdata <- Mdf[which(Mdf$istest == 0),]
testdata <- Mdf[which(Mdf$istest == 1),]


fasit_test <- testdata$y
fasit_training <- trainingdata$y

K = 30
feil_test = rep(NA, K)
feil_train = rep(NA, K)

for(k in 1:K){
  pred = class::knn(trainingdata, testdata, cl=trainingdata$y, k, l = 0, prob = FALSE, use.all = TRUE)
  feil_test[k] = mean(pred != fasit_test)
  pred_train = class::knn(trainingdata, trainingdata, cl=trainingdata$y, k, l = 0, prob = FALSE, use.a
ll = TRUE)
  feil_train[k] = mean(pred_train != fasit_training)
```

```
}

test.e = data.frame(k=1:K,error = feil_test)

ggplot(test.e,aes(x=k,y=error))+
  geom_point(col="red") +
  geom_line(linetype="dotted") +
  theme_minimal() +
  labs(title = "Misclassification error for the test data")
```




```
train.e = data.frame(k=1:K,error = feil_train)

ggplot(train.e,aes(x=k,y=error))+
  geom_point(col="red") +
  geom_line(linetype="dotted") +
  theme_minimal() +
  labs(title = "Misclassification error for the training data")
```



As expected, the error for the training data increases with k. For k=1 the classification is made to the same class as the one nearest neighbor, so the error will be small since the classifier is very flexible. For increasing k the error increases since the flexibility decreases.

However, for the test data the classifier is too flexible for low values of k. As a consequence the classifier will overfit the training data, resulting in a poor prediction for the test data. Therefore a very small k will give a large error for the test data.

Cross-validation

```
set.seed(0)
ks = 1:30 # Choose K from 1 to 30.
idx = createFolds(M[tr,1], k=5) # Divide the training data into 5 folds.
# "Sapply" is a more efficient for-loop.
# We loop over each fold and each value in "ks"
# and compute error rates for each combination.
# All the error rates are stored in the matrix "cv",
# where folds are rows and values of $K$ are columns.
cv = sapply(ks, function(k){
  sapply(seq_along(idx), function(j) {
    yhat = class::knn(train=M[tr[ -idx[[j]] ], -1],
                      cl=M[tr[ -idx[[j]] ], 1],
                      test=M[tr[ idx[[j]] ], -1], k = k)
    mean(M[tr[ idx[[j]] ], 1] != yhat)
  })
})
```

Q11: To compute the average $cv.e$ and standard error $cv.se$ of the average cross-validation error over all 5 folds we used the built-in functions seen below. The k corresponding to the smallest cross-validation error is stored in $k.min$

```
library(matrixStats)
cv.e = colMeans(cv)
cv.se = colSds(cv)
k.min = which.min(cv.e)
```

Q12:

```
library(colorspace)
co = rainbow_hcl(3)
par(mar=c(4,4,1,1)+.1, mgp = c(3, 1, 0))
plot(ks, cv.e, type="o", pch = 16, ylim = c(0, 0.7), col = co[2],
     xlab = "Number of neighbors", ylab="Misclassification error")
arrows(ks, cv.e-cv.se, ks, cv.e+cv.se, angle=90, length=.03, code=3, col=co[2])
lines(ks, train.e$error, type="o", pch = 16, ylim = c(0.5, 0.7), col = co[3])
lines(ks, test.e$error, type="o", pch = 16, ylim = c(0.5, 0.7), col = co[1])
legend("topright", legend = c("Test", "5-fold CV", "Training"), lty = 1, col=co)
```

The bias in $\hat{y}(x)$ increases with K since the flexibility decreases, causing the classifier to underfit. That is, the classifier can miss some relevant relations. On the contrary the variance will be high for low values of K. A too low value of K will give a very flexible classifier which will overfit the training set, giving a high variance. That is, the classifier can model random noise from the training data rather than the actual relations. This explaines the low misclassification error for the training set in the plot when K is small. Therefore, increasing K will decrease the flexibility, and then also decrease the variance.

Q13: Instead of choosing the K that results in the smallest CV error rate, a different strategy is implemented. In the first line of the code below the k is required to satisfy the inequality

$$cv.e < cv.e[k.min] + cv.se[k.min]$$

In words, the average cross validation for k has to be smaller than the smallest possible average found in **Q11** plus the standard deviation of the corresponding k.min.

The largest k that satisfies the inequality is set to be k. This will give a k that has both a small cross-validation average and standard deviation and a k that is hopefully large enough so it does not overfit the training data. In this case $k = 30$. A plot of this 30-nearest neighbors is shown below.

```
#the largest value of K that satisfies the ineq
k = tail(which(cv.e < cv.e[k.min] + cv.se[k.min]), 1)
size = 100
xnew = apply(M[tr,-1], 2, function(X) seq(min(X), max(X), length.out=size))
grid = expand.grid(xnew[,1], xnew[,2])
grid.yhat = class::knn(M[tr,-1], M[tr,1], k=k, test=grid)
np = 300
par(mar=rep(2,4), mgp = c(1, 1, 0))
contour(xnew[,1], xnew[,2], z = matrix(grid.yhat, size), levels=.5,
        xlab=expression("x"[1]), ylab=expression("x"[2]), axes=FALSE,
        main = paste0(k,"-nearest neighbors"), cex=1.2, labels="")
#abline(0,1) #Decision boundary for classifier in Q15
points(grid, pch=".", cex=1, col=grid.yhat)
points(M[1:np,-1], col=factor(M[1:np,1]), pch = 1, lwd = 1.5)
legend("topleft", c("Player 1 wins", "Player 2 wins"),
       col=c("red", "black"), pch=1)
box()
```

ROC and AUC

Q14: K is set to be the optimal choice of K found in **Q13**, so $K = 30$. The training set is used to produce the probability for player 1 winning the match for the matches in the test set. A ROC is produced based on the test set below.

```
K=30 # your choice from Q13

  # knn with prob=TRUE outputs the probability of the winning class
  # therefore we have to do an extra step to get the probability of player 1 winning
  KNNclass=class::knn(train=M[tr,-1], cl=M[tr,1], test=M[-tr,-1], k = K,prob=TRUE)
KNNprobwinning=attributes(KNNclass)$prob
KNNprob= ifelse(KNNclass == "0", 1-KNNprobwinning, KNNprobwinning)
# now KNNprob has probability that player 1 wins, for all matches in the test set
```

```
# now you use predictor=KNNprob and response=M[-tr,1]
# in your call to the function roc in the pROC library

library(pROC)
roc(response=M[-tr,1],predictor=KNNprob,plot= TRUE)
```

The ROC curve gives a graphical display of the sensitivity against specificity, for all possible threshold values. The sensitivity is the proportion of correctly classified positive observations, the true positive rate. The specificity is the proportion of correctly classified negative observations, the false positive rate. The thresold value is the cut-off on probability of success. A good classifier has both high sensitivity and high specificity. So an ideal classifier will give a ROC curve which hugs the top left corner. In this case the ROC curve plotted above shows this tendency, so the classifier is to some extent a good predictor for the classes.

The area under the ROC curve is found to be $AUC = 0.8178$. The AUC for a classifier ranges between 0 and 1, where a high value indicates a good classifier. This corresponds to the desired shape of the ROC cuve. If AUC = 0.5 it indicates that half of $\hat{y}$ is classified correct. Since there are two classes, this will give the same result as randomly guessing the class.

A competing classifier

$$\tilde{y}(x) = argmax_k(x_k)$$

The estimator $\tilde{y}$ is giving the win to the player with the highest quality score.

Q15:

```
N = 394                     #length of testdata
predicted = rep(NA, N)
for(i in 1:N){
  if (testdata$x1[i] > testdata$x2[i]){
    predicted[i] = 1
  }
  else {
    predicted[i] = 0
  }
}
predicted = as.factor(predicted)
cm <- confusionMatrix(predicted,testdata$y)
mcerror = 1 - cm$overall[1]
names(mcerror) <- "Missclassification error"
#cv.e[30]
```

For this exercise we have added a straight line in the plot in Q13, which illustrates the decision boundary for the classification method $\tilde{y} = argmax_k(x_k)$. All points below the line will be classified as wins for player 1, as the rating x_1 is greater than x_2 in this area. Similarily all points above the line will be classified as wins for player 2, as here the rating x_2 is greater than x_1.

Based on misclassificaion error we would preferably choose the $\tilde{y} = argmax_k(x_k)$ classifier, as the missclassification error is 0.2335, which is lower compared to 0.2848 with KNN classifier with K = 30.

Problem 3: Bias-variance trade-off

Let $\boldsymbol{x}$ be a $(p + 1) \times 1$ vector of covariates. A regression problem is given as

$$\boldsymbol{Y} = f(\boldsymbol{x}) + \epsilon, \text{ where } E[\epsilon] = 0 \text{ and } Var[\epsilon] = \sigma^2$$

The true function is assumed to be $f(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{\beta}$.

We will consider two estimators $\hat{\boldsymbol{\beta}}$ and $\tilde{\boldsymbol{\beta}}$ for $\boldsymbol{\beta}$, and let the prediction of a new response at some covariate vector $\boldsymbol{x}_0$ be

$$\hat{f}(\boldsymbol{x}_0) = \boldsymbol{x}_0^T\hat{\boldsymbol{\beta}} \text{ or } \tilde{f}(\boldsymbol{x}_0) = \boldsymbol{x}_0^T\tilde{\boldsymbol{\beta}}$$

The estimators are based on a training set, where $\boldsymbol{X}$ is a n×(p−1) design matrix and $boldsymbolY$ is a n×1 vector of reponses, where we assume that the observed responses are independent of each other. Another assumption is that $E[\boldsymbol{Y}] = \boldsymbol{X\beta}$ and $Cov[\boldsymbol{Y}] = \sigma^2\boldsymbol{I}$.

Classical least squares estimator

We will first consider the leat square estimator

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y}$$

Q16: For a design matrix $\boldsymbol{X}$ and a vector of response the following holds

$$E[\boldsymbol{XY}] = \boldsymbol{X}E[\boldsymbol{Y}] \text{ and } Cov[\boldsymbol{XY}] = \boldsymbol{X}Cov[\boldsymbol{Y}]\boldsymbol{X}^T.$$

The expected value vector for $\hat{\boldsymbol{\beta}}$ is therefore given as

$$E[\hat{\boldsymbol{\beta}}] = E[(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y}] = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^TE[\boldsymbol{Y}] = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\beta} = \boldsymbol{\beta}$$

and the variance-covariance matrix for $\hat{\boldsymbol{\beta}}$ is given as

$$Cov[\hat{\boldsymbol{\beta}}] = Cov[(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y}] = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^TCov[\boldsymbol{Y}]((\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T)^T = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{I}\sigma^2((\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T)^T$$

$$= \sigma^2(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{X}((\boldsymbol{X}^T\boldsymbol{X})^{-1})^T = \sigma^2(\boldsymbol{X}^T\boldsymbol{X})^{-1}$$

Q17: The expected value and variance of $\hat{f}(\boldsymbol{x}_0)$ are then given as

$$E[\hat{f}(\boldsymbol{x}_0)] = E[\boldsymbol{x}_0^T\hat{\boldsymbol{\beta}}] = \boldsymbol{x}_0^TE[\hat{\boldsymbol{\beta}}] = \boldsymbol{x}_0^T\boldsymbol{\beta}$$

$$Var[\hat{f}(\boldsymbol{x}_0)] = Cov[\boldsymbol{x}_0^T\hat{\boldsymbol{\beta}}] = \boldsymbol{x}_0^TCov[\hat{\boldsymbol{\beta}}]\boldsymbol{x}_0 = \sigma^2\boldsymbol{x}_0^T(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{x}_0$$

Q18: With the information from **Q17** and the fact that $Var[\boldsymbol{Y}_0] = E[\boldsymbol{Y}_0^2] - E[\boldsymbol{Y}_0]^2$, $\mathrm{E}[(\boldsymbol{Y}_0 - \hat{f}(\mathbf{x}_0))^2]$ can be written as

$$\mathrm{E}[(\boldsymbol{Y}_0 - \hat{f}(\mathbf{x}_0))^2] = \mathrm{E}[\boldsymbol{Y}_0^2 - 2\boldsymbol{Y}_0\hat{f}(\mathbf{x}_0) + \hat{f}(\mathbf{x}_0)^2] = \mathrm{E}[\boldsymbol{Y}_0^2] - 2\mathrm{E}[\boldsymbol{Y}_0]\mathrm{E}[\hat{f}(\mathbf{x}_0)] + \mathrm{E}[\hat{f}(\mathbf{x}_0)^2] = Var[\boldsymbol{Y}_0] + \mathrm{E}[\boldsymbol{Y}_0]^2 - 2\mathrm{E}[\boldsymbol{Y}_0]\mathrm{E}[\hat{f}(\mathbf{x}_0)] + Var[\hat{f}(\mathbf{x}_0)] + \mathrm{E}[\hat{f}(\mathbf{x}_0)]^2$$

Furthermore, $E[\boldsymbol{Y}_0] = f(\mathbf{x}_0)$ and $Var[\boldsymbol{Y}_0] = Var(\epsilon)$, which leads to

$$f(\mathbf{x}_0)^2 - 2f(\mathbf{x}_0)\mathrm{E}[\hat{f}(\mathbf{x}_0)] + \mathrm{E}[\hat{f}(\mathbf{x}_0)]^2 + Var[\hat{f}(\mathbf{x}_0)] + Var[\epsilon] = (\mathrm{E}[\hat{f}(\mathbf{x}_0)] - f(\mathbf{x}_0))^2 + \mathrm{Var}[\hat{f}(\mathbf{x}_0)] + \mathrm{Var}[\varepsilon]$$

The expression is now a sum of the squared bias $(\mathrm{E}[\hat{f}(\mathbf{x}_0) - f(\mathbf{x}_0)]^2$, the variance $\mathrm{Var}[\hat{f}(\mathbf{x}_0)]$ and the irreducible error $\mathrm{Var}[\varepsilon]$.

Inserting the results found in the two previous questions this can further be expressed as

$$(\boldsymbol{x}_0^T\beta - f(\mathbf{x}_0))^2 + \sigma^2\boldsymbol{x}_0^T(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{x}_0 + \mathrm{Var}[\varepsilon]$$

Ridge regression estimator

We will now consider a competing estimator given as

$$\tilde{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y}$$

for some choice of a numerical parameter λ.

Q19: The expected value is given by:

$$E[\tilde{\boldsymbol{\beta}}] = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T \cdot E[\mathbf{Y}] = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}$$

As we can see, for $\lambda = 0$ our estimater is unbiased.

The variance-covariance matrix for $\tilde{\boldsymbol{\beta}}$ is given by:

$$Var[\tilde{\boldsymbol{\beta}}] = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T \cdot Cov[\mathbf{Y}] \cdot \mathbf{X}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1} = \sigma^2(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{X}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}$$

Q20: To find the expected value and variance of $\tilde{f}(x_0) = \mathbf{x_0^T}\tilde{\beta}$ we write:

$$E[\tilde{f}(\mathbf{x_0})] = E[\mathbf{x_0^T}\tilde{\boldsymbol{\beta}}] = \mathbf{x_0^T}E[\tilde{\boldsymbol{\beta}}] = \mathbf{x_0^T}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{X}\boldsymbol{\beta}$$

$$Var[\tilde{f}(\mathbf{x_0})] = Var[\mathbf{x_0^T}\tilde{\boldsymbol{\beta}}] = \mathbf{x_0^T}Var[\tilde{\boldsymbol{\beta}}]\mathbf{x_0} = \sigma^2\mathbf{x_0^T}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{X}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{x_0}$$

Q21: $\mathrm{E}[(\boldsymbol{Y}_0 - \tilde{f}(\mathbf{x}_0))^2]$ can be obtained by the same procidure as in **Q18**, by substituting $\hat{\boldsymbol{\beta}}$ with $\tilde{\boldsymbol{\beta}}$ and $\hat{f}(\boldsymbol{\beta})$ with $\tilde{f}(\boldsymbol{\beta})$. This leads to the expression

$$\mathrm{E}[(\boldsymbol{Y}_0 - \tilde{f}(\mathbf{x}_0))^2] = (\mathrm{E}[\tilde{f}(\mathbf{x}_0)] - f(\mathbf{x}_0))^2 + \mathrm{Var}(\tilde{f}(\mathbf{x}_0)) + \mathrm{Var}(\varepsilon)$$

Inserting the results found in the two previous questions this can further be expressed as

$$\mathrm{E}[(\boldsymbol{Y}_0 - \tilde{f}(\mathbf{x}_0))^2] = (\mathbf{x_0^T}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} - f(\mathbf{x}_0))^2 + \sigma^2\mathbf{x_0^T}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{X}(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{x_0} + \mathrm{Var}(\varepsilon)$$

Plotting the three components

```
values=dget("https://www.math.ntnu.no/emner/TMA4268/2019v/data/BVtradeoffvalues.dd")
X=values$X
dim(X)
x0=values$x0
dim(x0)
beta=values$beta
dim(beta)
sigma=values$sigma
sigma
```

```
## [1] 100  81
## [1] 81  1
## [1] 81  1
## [1] 0.5
```

Q22: The squared bias is plotted as a function of λ for one set of values for X, x_0, β and σ below.

```
sqbias=function(lambda,X,x0,beta)
{
  p=dim(X)[2]
  value= (t(x0)%*%solve(t(X)%*%X+lambda*diag(p))%*%(t(X)%*%X)%*%beta-(t(x0)%*%beta))^2
  return(value)
}
thislambda=seq(0,2,length=500)
sqbiaslambda=rep(NA,length(thislambda))
for (i in 1:length(thislambda)) sqbiaslambda[i]=sqbias(thislambda[i],X,x0,beta)
plot(thislambda,sqbiaslambda,col=2,type="l")
```

We expect the squared bias to increase as λ increases. The reason for this is that the expected value of $\tilde{\beta}$ increases as λ grows larger, causing the expected value of $\tilde{f}(x_0)$ to also grow with λ. Hence the difference between the expected value of $\tilde{f}(x_0)$ and the actual value of $f(x_0)$ increases, which is the bias.

Q23: The variance is plotted as a function of λ for one set of values for X, x_0, β and σ below.

```
variance=function(lambda,X,x0,sigma)
{
  p=dim(X)[2]
  inv=solve(t(X)%*%X+lambda*diag(p))
  value=sigma^2*(t(x0)%*%(inv)%*%t(X)%*%X%*%(inv)%*%x0)
    return(value)
}
thislambda=seq(0,2,length=500)
variancelambda=rep(NA,length(thislambda))
```

```
for (i in 1:length(thislambda)) variancelambda[i]=variance(thislambda[i],X,x0,sigma)
plot(thislambda,variancelambda,col=4,type="l")
```

On the contrary to the bias, we expect the variance to decrease as λ increases. We expect this beacuse as λ grows larger the variance of $\tilde{f}(x_0)$ decreases, since the term $(X^TX + \lambda I)$ has the exponent -1. One can also predict this decrease because of the bias-variance trade-off, which has the property that lower bias in parameter estimation have a higher variance and vice versa. From the plot this is indeed the case.

Q24: Now the squared bias, the variance and the irreducible error is plotted together with the sum of the three as a function of λ.

```
tot=sqbiaslambda+variancelambda+sigma^2
which.min(tot)
thislambda[which.min(tot)]
plot(thislambda,tot,col=1,type="l",ylim=c(0,max(tot)))
lines(thislambda, sqbiaslambda,col=2)
lines(thislambda, variancelambda,col=4)
lines(thislambda,rep(sigma^2,500),col="orange")
abline(v=thislambda[which.min(tot)],col=3)
v=thislambda[which.min(tot)]
```

The optimal value of λ for this problem is when the sum of the squared bias, the variance and the irreducible error (the black curve) is smallest. In this case that is when $\lambda = 0.993988$.