Compulsory exercise 1: Group 10

TMA4268 Statistical Learning V2019

Martin Kvisvik Larsen

22 februar, 2019

Problem 1: Multiple linear regression

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

```
##
## Call:
## lm(formula = log(FEV) ~ Age + Htcm + Gender + Smoke, data = lungcap)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.63278 -0.08657  0.01146  0.09540  0.40701
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.943998   0.078639 -24.721  < 2e-16 ***
## Age          0.023387   0.003348   6.984  7.1e-12 ***
## Htcm         0.016849   0.000661  25.489  < 2e-16 ***
## GenderM      0.029319   0.011719   2.502   0.0126 *
## Smoke       -0.046067   0.020910  -2.203   0.0279 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1455 on 649 degrees of freedom
## Multiple R-squared:  0.8106, Adjusted R-squared:  0.8095
## F-statistic: 694.6 on 4 and 649 DF,  p-value: < 2.2e-16
```

Q1:

Model A can be written as:

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \beta_4 x_{i4} + \varepsilon, \qquad i = 1, ..., n$$

where Y_i is the logarithm of the FEV, x_{i1} is the age, x_{i2} is the htcm, x_{i3} is the gender and x_{i4} is the smoke of sample i and ε is normally distributed noise with 0 mean and variance σ^2. The model can be written in matrix form as:

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} & x_{13} & x_{14} \\ 1 & x_{21} & x_{22} & x_{23} & x_{24} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} & x_{n3} & x_{n4} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix} + \begin{bmatrix} \varepsilon \\ \varepsilon \\ \vdots \\ \varepsilon \end{bmatrix}$$

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

Q2:

`Estimate` are the values of the regression coefficients estimates $\hat{\beta}_j$. When using the least squares estimators the coefficients are calculated as follows:

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

`Intercept` is the value of $\hat{\beta}_0$, which is the average value of Y when all the covariates x_{ij} are 0.

`Std.Error` is the standard error of the regression coefficient estimates, i.e. how much they vary depending on the covariates x_i and the variance of the noise ε. In simple linear regression, when assuming that the regression coefficients $\hat{\beta}_0$ and $\hat{\beta}_1$ are uncorrelated the standard error of the two coefficients are given as:

$$\text{SE}(\hat{\beta}_0) = \sqrt{\sigma^2\left(\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}\right)}$$

$$\text{SE}(\hat{\beta}_1) = \sqrt{\frac{\sigma^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}}$$

`Residual standard error` is a measurement of the deviation between the measured reponse Y and the estimated regression line $\hat{Y}$ of our estimator. The residual standard error is given by:

$$\text{RSE} = \sqrt{\frac{1}{n-2}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}$$

`F-statistic` is a statistic that is used to test if the regression is significant, i.e. that atlest one of the regression parameters of the model, β_j are different from zero. A large F-statistic means that one can be confident that there is correlation between one of the covariates and the response. The F-statistic is used in the following hypothesis test for testing regression significance:

$$H_0 : \beta_0 = \beta_1 = \cdots = \beta_p = 0 \quad \text{vs.} \quad H_1 : \text{at least one } \beta_j \text{ different from zero}$$

In this hypothesis test the F-statistic is defined as:

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

f_0 is the numerical value of F when the RSS and TSS from the data have been inserted. The p-value of the hypothesis test is then given as p-value $= P(F_{p,n-p-1} > f_0)$, where n is the number of data samples and p is the number of regression parameters.

Q3:

```r
R_squared_A = summary(modelA)$r.squared
```

Proportion of variability explained by the model can be measured by the coefficient of determination R^2, defined as:

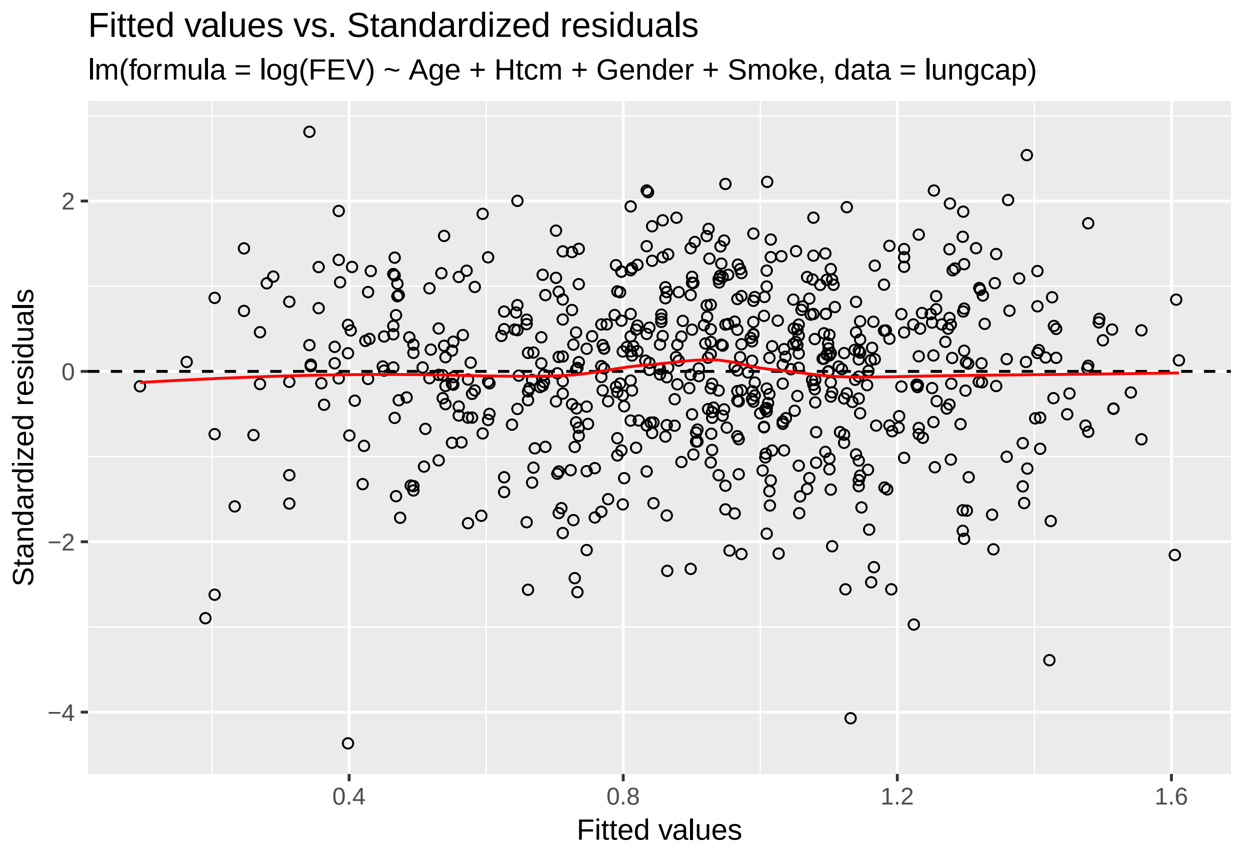
$$R^2 = \frac{\text{TSS} - \text{RSS}}{\text{TSS}} = 1 - \frac{\text{RSS}}{\text{TSS}} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}$$

The coefficient of determination for model A is: 0.8106

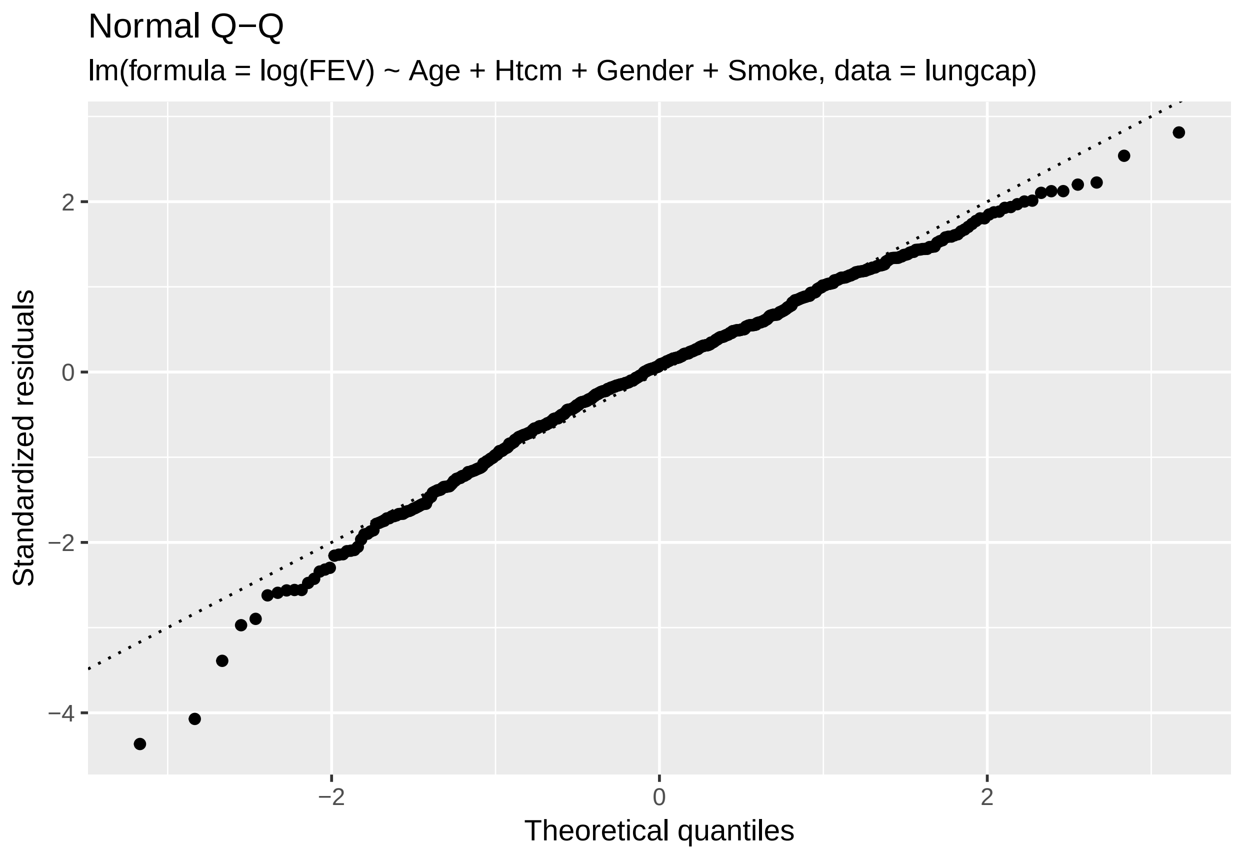
The coefficient of determination of model A is reasonably large, seeing as a model that would fit the data perfectly would have a coefficient of determination of 1. However, the coefficient of determination cannot be reduced by adding more covariates, hence it favours more complex models.

Q4:

```r
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))
```



```r
# 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))
```



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

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

Fitted values vs. standardized residuals: From the fitted values vs. standarized residuals one can see that the spread around the horizontal line is close to constant, but that there is a clear pattern around the fitted values interval [0.8, 1]. The pattern indicates that there is a non-linear relationship between the response and the covariates, while the constant spread indicates that the error variances is somewhat constant.

Normal Q-Q plot: From the normal Q-Q plot one can see that the residuals do not lie on straight line. This indicates that the residuals are not normally distributed, which indicates that the difference between the model and the measurements is not only normally distributed noise.

All in all the model is ok, but not great due to the indicated non-linear response-covariate relationship and the non-normally distributed residuals.

Q5:

```
# here you write your code
modelB = lm(FEV ~ Age + Htcm + Gender + Smoke, data=lungcap)
summary(modelB)

R_squared_B = summary(modelB)$r.squared
R_squared_adj_A = summary(modelA)$adj.r.squared
R_squared_adj_B = summary(modelB)$adj.r.squared
```

```
##
## Call:
## lm(formula = FEV ~ Age + Htcm + Gender + Smoke, data = lungcap)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.37656 -0.25033  0.00894  0.25588  1.92047
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -4.456974   0.222839 -20.001  < 2e-16 ***
## Age          0.065509   0.009489   6.904 1.21e-11 ***
## Htcm         0.041023   0.001873  21.901  < 2e-16 ***
## GenderM      0.157103   0.033207   4.731 2.74e-06 ***
## Smoke       -0.087246   0.059254  -1.472    0.141
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4122 on 649 degrees of freedom
## Multiple R-squared:  0.7754, Adjusted R-squared:  0.774
## F-statistic:   560 on 4 and 649 DF,  p-value: < 2.2e-16
```

From the summary of model A and the summary of model B one can see that model A has a R^2_{adj} value of 0.8095, while model B has an adjusted R^2_{adj} value of 0.774. Since model A has a larger R^2_{adj} value than model B, the model of choice is A.

Q6:

```
# here you write your code if you have any
beta_age = summary(modelA)$coefficients["Age",1]
p_value_age = summary(modelA)$coefficients["Age",4]
```

To test if Age has an effect on log(FEV) one can perform the following hypothesis test,

$$H_0 : \beta_{\text{Age}} = 0 \qquad\qquad H_1 : \beta_{\text{Age}} \neq 0$$

by using the T-statistic:

$$T_{\text{Age}} = \frac{\hat{\beta}_{\text{Age}} - \beta_{\text{Age}}}{\sqrt{c_{\text{AgeAge}}}\hat{\sigma}}$$

Then, the p-value can be calculated as:

$$\text{p-value} = P(t_{n-p-1} > T_{\text{Age}})$$

The p-value for the hypothesis test is: 7.096e-12

The null hypothesis can be rejected at significance level: 7.096e-12

Q7:

```
beta_age_conf_int = confint(modelA, "Age", 0.95)
```

For the given significance level α the probability

$$P(\hat{\beta}_j - t_{\alpha/2,n-p-1}\sqrt{c_{jj}}\hat{\sigma} \leq \beta_j \leq \hat{\beta}_j + t_{\alpha/2,n-p-1}\sqrt{c_{jj}}\hat{\sigma}) = 1 - \alpha$$

gives the $(1-\alpha)\%$ confidence interval $[\hat{\beta}_j - t_{\alpha/2,n-p-1}\sqrt{c_{jj}}\hat{\sigma}, \hat{\beta}_j + t_{\alpha/2,n-p-1}\sqrt{c_{jj}}\hat{\sigma}]$. The interval means that one can with $(1-\alpha)\%$ confidence say that the true coefficient β_j lies within the interval. For β_{Age} the $(1-0.05)\%$ confidence interval is: [0.01681, 0.02996]

Since the null hypothesis was rejected at p-value $=$ 7.096e-12 and $\alpha = 0.05 >$ p-value $=$ 7.096e-12, the $(1-0.05)\%$ confidence interval will not contain the β_{Age}-value of the null hypothesis, i.e. $\beta_{\text{Age}} = 0$.

Q8:

```
new = data.frame(Age=16, Htcm=170, Gender="M", Smoke=0)
modelA_log_pred_int = predict(modelA, newdata=new, interval="prediction", type="response", level=0.95)
modelA_best_pred = exp(modelA_log_pred_int[1])
modelA_pred_low = exp(modelA_log_pred_int[2])
modelA_pred_high = exp(modelA_log_pred_int[3])
```

The best prediction for the 16 year old male's FEV is: 3.758

The (0.95)% prediction interval for the 16 year old male's FEV is: [2.818, 5.01]

It is useful to be able to get a point prediction and quantify the upper and lower bounds of the prediction (at a certain significance level). A single point prediction without the prediction interval is not useful at all as it does not provide the user any information about the uncertainty of the model. If such a prediction would be the basis for decision making, the user would not know if the decision was made on solid or shaky ground.

Problem 2: Classification

```
library(class)# for function knn
library(caret)# for confusion matrices
```

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

```
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:

The equation for the KNN estmator is:

$$\hat{y}(x) = \text{argmax}_j(\hat{P}(Y = j|X = x))$$

The estimated conditional probabilities are calculated as:

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

Q10:

```
ks = 1:30
train.e = rep(NA,length(ks))
test.e = rep(NA,length(ks))
train = M[tr,-1]
test = M[-tr,-1]
train_labels = M[tr,1]
test_labels = M[-tr,1]

for (k in ks){
  y_hat_test = class::knn(train=train, cl=train_labels, test=test, k=k)
  y_hat_train = class::knn(train=train, cl=train_labels, test=train, k=k)
  test.e[k] = mean(test_labels != y_hat_test)
  train.e[k] = mean(train_labels != y_hat_train)
}
```

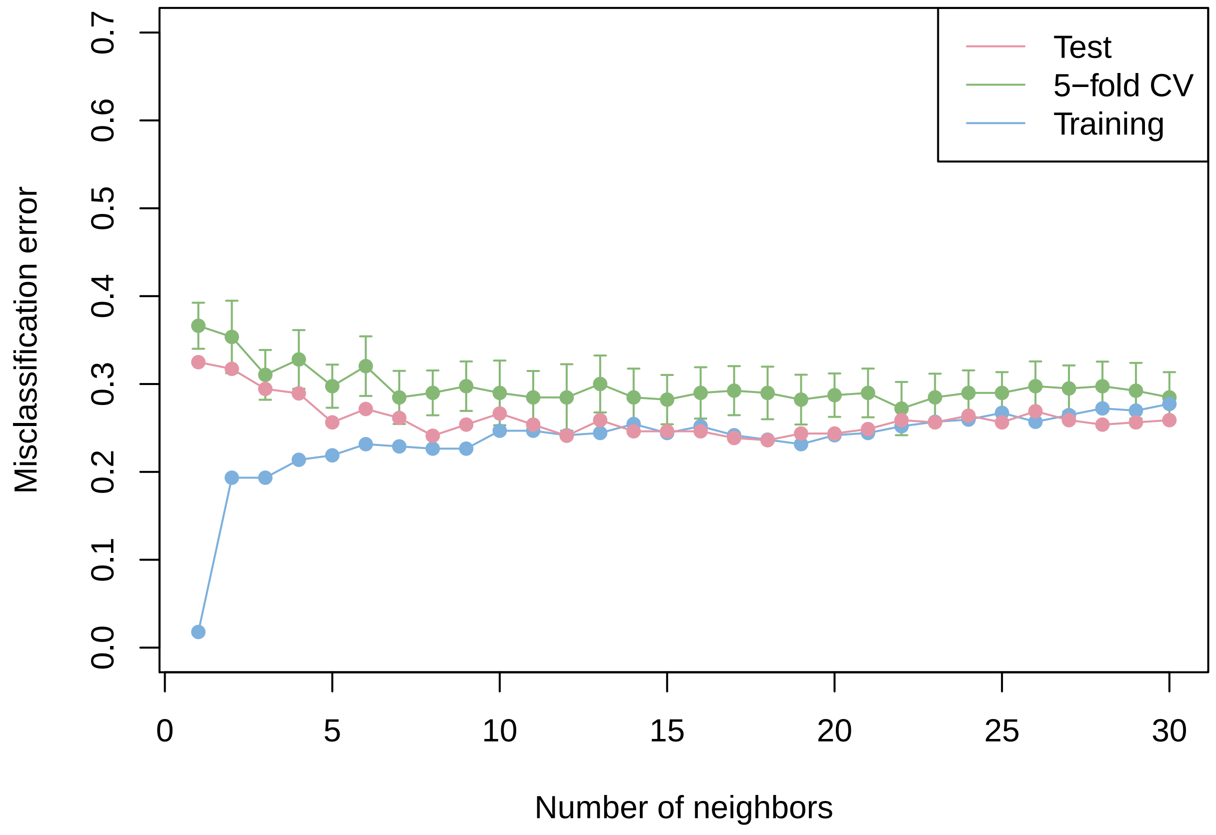
```
set.seed(0)
ks = 1:30 # Choose K from 1 to 30.
idx = createFolds(M[tr,1], k=5) # Divide the training data into 5 folds.
# "Sapply" is a more efficient for-loop.
# We loop over each fold and each value in "ks"
# and compute error rates for each combination.
# All the error rates are stored in the 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:

```
cv.e = colMeans(cv)
n_folds = dim(cv)[1]
cv.se = (1/sqrt(n_folds)) * apply(cv, 2, sd)
k.min = which.min(cv.e)
```

Q12:

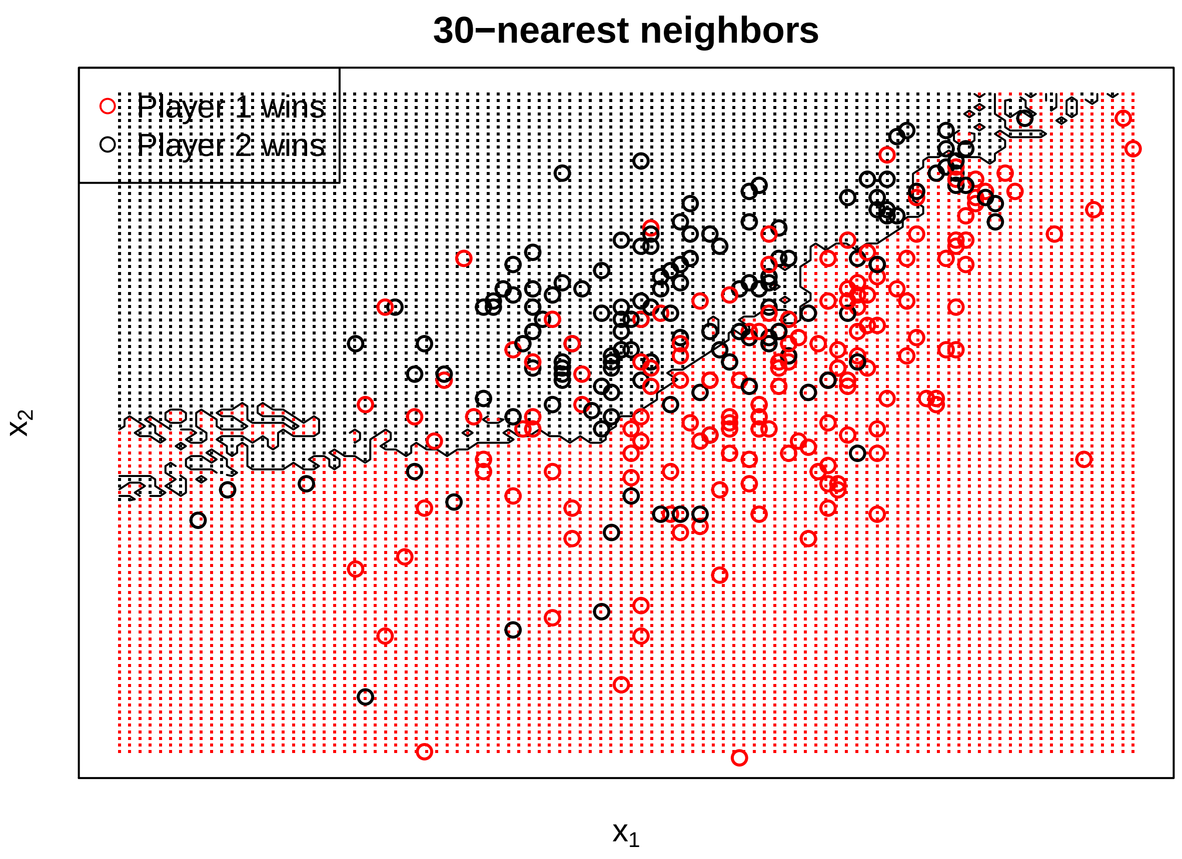
```
library(colorspace)
co = rainbow_hcl(3)
par(mar=c(4,4,1,1)+.1, mgp = c(3, 1, 0))
plot(ks, cv.e, type="o", pch = 16, ylim = c(0, 0.7), col = co[2],
     xlab = "Number of neighbors", ylab="Misclassification error")
arrows(ks, cv.e-cv.se, ks, cv.e+cv.se, angle=90, length=.03, code=3, col=co[2])
lines(ks, train.e, type="o", pch = 16, ylim = c(0.5, 0.7), col = co[3])
lines(ks, test.e, 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 $\hat{y}(x)$ will increase with K. The most clear example of this in the plot is the misclassification error on the training set, which for $K = 1$ is close to 0, but jumps to about 0.2 for $K = 2$ and then tends to continue to increase for increasing values of K. The variance in $\hat{y}(x)$ decreases with K. In the plot one can see this from the standard error of the 5-fold cross validation (although the scale of the y-axis makes it a bit hard). The standard error (and thus the variance of $\hat{y}(x)$) do in general become smaller for increasing values of K.

Q13:

```
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 = 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="")
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()
```



The proposed strategy is to use the one standard error rule. Denoting the model with the smallest cross-validation error $\hat{\theta}$, that is:

$$\hat{\theta} = argmin_{\theta}\text{CV}(\theta)$$

The one standard error rule is then to choose the simplest model, i.e. the largest value of K in the case of the KNN-classifier, that satisfies the following inequality:

$$\text{CV}(\theta) \leq \text{CV}(\hat{\theta}) + \text{SE}(\hat{\theta})$$

By using the one standard error rule one picks more simple models, which hopefully generalizes better than the model with the smallest cross-validation error.

Q14:

```
K=30

# knn with prob=TRUE outputs the probability of the winning class
```

```r
# 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

library(pROC)
```

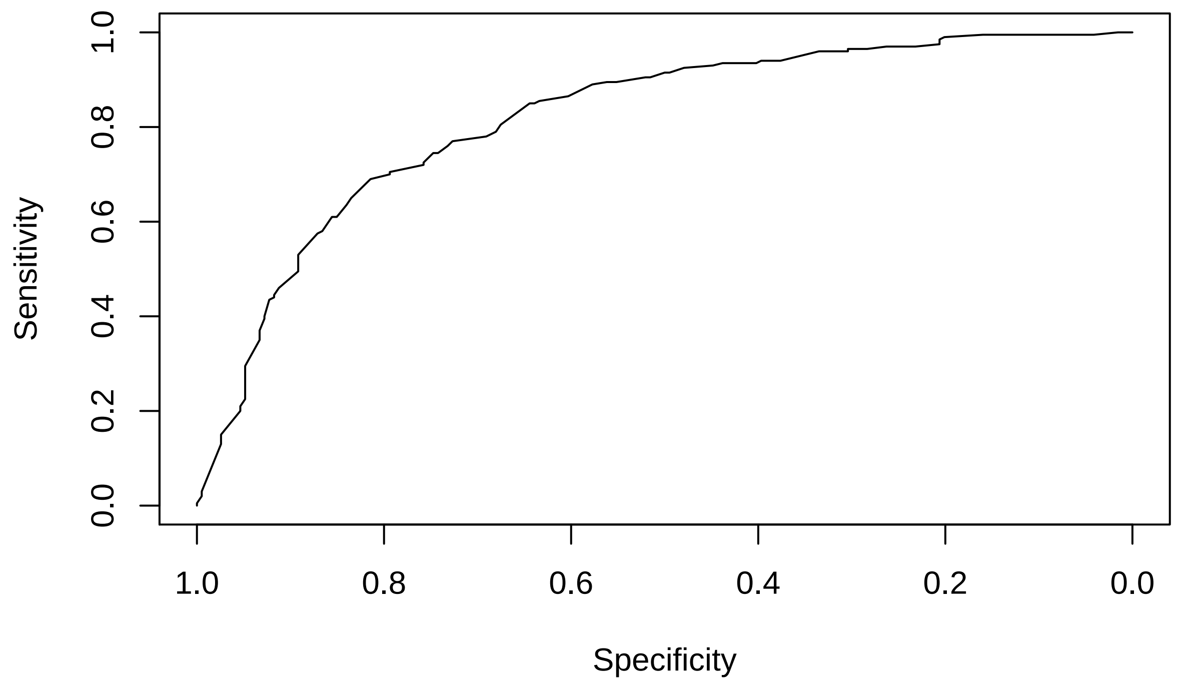
```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following object is masked from 'package:colorspace':
##
##     coords

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
# now you use predictor=KNNprob and response=M[-tr,1]
# in your call to the function roc in the pROC library
my_roc = roc(response=M[-tr,1], predictor=KNNprob)
plot(my_roc$specificities, my_roc$sensitivities, type="l",
     xlim=c(1.0, 0), ylim=c(0, 1.0), xlab = "Specificity", ylab="Sensitivity")
```



The ROC is produced by calculating the sensitivity and specificity of the classifier for all possible values of the threshold value, i.e. the cutoff probability of success or disease. Sensitivity and specificity are defined as follows:

$$\text{Sensitivity} = \frac{\text{number of true positives}}{\text{number of positives}} = \frac{\text{number of true positives}}{\text{number of true positives} + \text{number of false negatives}}$$

$$\text{Specificity} = \frac{\text{number of true negatives}}{\text{number of negatives}} = \frac{\text{number of true negatives}}{\text{number of true negatives} + \text{number of false positives}}$$

The area under the ROC curve (AUC) is: 0.8178

When randomly guessing the chance of guessing success or failure is just given by the threshold value. When the threshold value is 1 all guess are gonna be negative and hence the number of false positives and true positives are going to be 0. In other word the sensitivity is 0 and the specificity is 1. It is the other way around when the threshold value is 0. Generally the ratio between negative and positive guesses is linear with the threshold value, which is why randomly guessing produces a line from $(1.0, 0)$ to $(0, 1.0)$ in a ROC curve. When this line is integrated it yields an AOC of 0.5.

Q15:

```r
y_bar = function(x){
  n_vars = dim(x)[2]
  n_samps = dim(x)[1]

  y_bar = rep(NA,n_samps)

  for(i in 1:n_samps){
    max_idx = which.max(x[i,])
    y_bar[i] = max_idx - 1
  }
  y_bar = factor(y_bar)
  return(y_bar)
}

y_bar_test = y_bar(test)
y_hat_test = class::knn(train=train, cl=train_labels, test=test, k=K)
y_bar_error = mean(test_labels != y_bar_test)
y_hat_error = mean(test_labels != y_hat_test)

confusionMatrix(y_bar_test, test_labels)

confusionMatrix(y_hat_test, test_labels)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  49 157
##          1 145  43
##
##                Accuracy : 0.2335
##                  95% CI : (0.1926, 0.2785)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : 1.0000
##
##                   Kappa : -0.5319
##  Mcnemar's Test P-Value : 0.5267
##
##             Sensitivity : 0.2526
##             Specificity : 0.2150
##          Pos Pred Value : 0.2379
##          Neg Pred Value : 0.2287
##              Prevalence : 0.4924
##          Detection Rate : 0.1244
##    Detection Prevalence : 0.5228
##       Balanced Accuracy : 0.2338
##
##        'Positive' Class : 0
##
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 138  44
##          1  56 156
##
##                Accuracy : 0.7462
##                  95% CI : (0.7002, 0.7884)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4918
##  Mcnemar's Test P-Value : 0.2713
##
##             Sensitivity : 0.7113
##             Specificity : 0.7800
##          Pos Pred Value : 0.7582
##          Neg Pred Value : 0.7358
##              Prevalence : 0.4924
##          Detection Rate : 0.3503
##    Detection Prevalence : 0.4619
##       Balanced Accuracy : 0.7457
##
##        'Positive' Class : 0
##
```

Misclassification error on the test set of $\bar{y}(x)$: 0.7665

Misclassification error on the test set of $\hat{y}(x)$ with $K = 30$: 0.2538

I would pick the KNN-classifier $\hat{y}(x)$.

Problem 3: Bias-variance trade-off

Q16:

$$\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{\beta}}] &= \mathrm{E}[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}] = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathrm{E}[\mathbf{Y}] \\
&= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathrm{E}[\mathbf{X}\boldsymbol{\beta}+\boldsymbol{\epsilon}] = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T(\mathrm{E}[\mathbf{X}\boldsymbol{\beta}]+\mathrm{E}[\boldsymbol{\epsilon}]) \\
&= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathrm{E}[\mathbf{X}\boldsymbol{\beta}] = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} \\
&= \boldsymbol{\beta}
\end{aligned}$$

$$\begin{aligned}
\mathrm{Cov}[\hat{\boldsymbol{\beta}}] &= \mathrm{Cov}[(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}] = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathrm{Cov}[\mathbf{Y}]((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T)^T \\
&= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\sigma^2\mathbf{I}((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T)^T = \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{I}((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T)^T \\
&= \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}((\mathbf{X}^T\mathbf{X})^{-1})^T = \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}((\mathbf{X}^T\mathbf{X})^T)^{-1} \\
&= \sigma^2(\mathbf{X}^T\mathbf{X})^{-1}
\end{aligned}$$

Q17:

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

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

Q18:

$$\begin{aligned}
\mathrm{E}[(Y_0-\hat{f}(\mathbf{x}_0))^2] &= \mathrm{E}[\hat{f}(\mathbf{x}_0)-f(\mathbf{x}_0)]^2 + \mathrm{Var}[\hat{f}(\mathbf{x}_0)] + \mathrm{Var}[\varepsilon] \\
&= \mathrm{E}[\mathbf{x}_0^T\hat{\boldsymbol{\beta}} - \mathbf{x}_0^T\boldsymbol{\beta}]^2 + \sigma^2\mathbf{x}_0^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_0 + \sigma^2 \\
&= \mathrm{E}[\mathbf{x}_0^T(\hat{\boldsymbol{\beta}}-\boldsymbol{\beta})]^2 + \sigma^2\mathbf{x}_0^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_0 + \sigma^2 \\
&= (\mathbf{x}_0^T\mathrm{E}[\hat{\boldsymbol{\beta}}-\boldsymbol{\beta}])^2\mathbf{x}_0 + \sigma^2\mathbf{x}_0^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_0) + \sigma^2 \\
&= (\mathbf{x}_0^T(\mathrm{E}[\hat{\boldsymbol{\beta}}]-\mathrm{E}[\boldsymbol{\beta}]))^2 + \sigma^2\mathbf{x}_0^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_0) + \sigma^2 \\
&= (\mathbf{x}_0^T(\boldsymbol{\beta}-\boldsymbol{\beta}))^2 + \sigma^2\mathbf{x}_0^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_0) + \sigma^2 \\
&= \sigma^2\mathbf{x}_0^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_0) + \sigma^2
\end{aligned}$$

Ridge estimator:

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

Q19:

Introducing the following matrix to make the expressions cleaner:

$$\mathbf{W} = (\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T$$

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

$$\begin{aligned}
\mathrm{Cov}[\widetilde{\boldsymbol{\beta}}] &= \mathrm{Cov}[(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y}] = (\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathrm{Cov}[\mathbf{Y}]((\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T)^T \\
&= (\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T\sigma^2\mathbf{I}((\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T)^T = \sigma^2(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{X}((\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^T)^{-1} \\
&= \sigma^2(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{X}(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1} \\
&= \sigma^2\mathbf{W}\mathbf{W}^T
\end{aligned}$$

Q20:

$$\mathrm{E}[\widetilde{f}(\mathbf{x}_0)] = \mathrm{E}[\mathbf{x}_0^T\widetilde{\boldsymbol\beta}] = \mathbf{x}_0^T\mathrm{E}[\widetilde{\boldsymbol\beta}] = \mathbf{x}_0^T(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{X}\boldsymbol\beta = \mathbf{x}_0^T\mathbf{W}\mathbf{X}\boldsymbol\beta$$

$$\begin{aligned}\mathrm{Var}[\widetilde{f}(\mathbf{x}_0)] &= \mathrm{Var}[\mathbf{x}_0^T\widetilde{\boldsymbol\beta}] = \mathbf{x}_0^T\mathrm{Cov}[\widetilde{\boldsymbol\beta}]\mathbf{x}_0\\ &= \sigma^2\mathbf{x}_0^T(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{X}(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{x}_0\\ &= \sigma^2\mathbf{x}_0^T\mathbf{W}\mathbf{W}^T\mathbf{x}_0\end{aligned}$$

Q21:

$$\begin{aligned}\mathrm{E}[(Y_0-\widetilde{f}(\mathbf{x}_0))^2] &= \mathrm{E}[\widetilde{f}(\mathbf{x}_0)-f(\mathbf{x}_0)]^2+\mathrm{Var}(\widetilde{f}(\mathbf{x}_0))+\mathrm{Var}(\varepsilon)\\ &= \mathrm{E}[\mathbf{x}_0^T\widetilde{\boldsymbol\beta}-\mathbf{x}_0^T\boldsymbol\beta]^2+\sigma^2\mathbf{x}_0^T(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{X}(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{x}_0+\sigma^2\\ &= (\mathbf{x}_0^T\mathrm{E}[\widetilde{\boldsymbol\beta}-\boldsymbol\beta])^2+\sigma^2\mathbf{x}_0^T(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{X}(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{x}_0)+\sigma^2\\ &= (\mathbf{x}_0^T(\mathrm{E}[\widetilde{\boldsymbol\beta}]-\mathrm{E}[\boldsymbol\beta]))^2+\sigma^2\mathbf{x}_0^T(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{X}(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{x}_0)+\sigma^2\\ &= (\mathbf{x}_0^T((\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{X}\boldsymbol\beta-\boldsymbol\beta))^2+\sigma^2\mathbf{x}_0^T(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{X}(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{x}_0)+\sigma^2\\ &= (\mathbf{x}_0^T((\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{X}-\mathbf{I})\boldsymbol\beta)^2+\sigma^2\mathbf{x}_0^T(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{X}(\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\mathbf{x}_0)+\sigma^2\\ &= (\mathbf{x}_0^T(\mathbf{W}\mathbf{X}-\mathbf{I})\boldsymbol\beta)^2+\sigma^2\mathbf{x}_0^T\mathbf{W}\mathbf{W}^T\mathbf{x}_0+\sigma^2\end{aligned}$$

```r
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
```

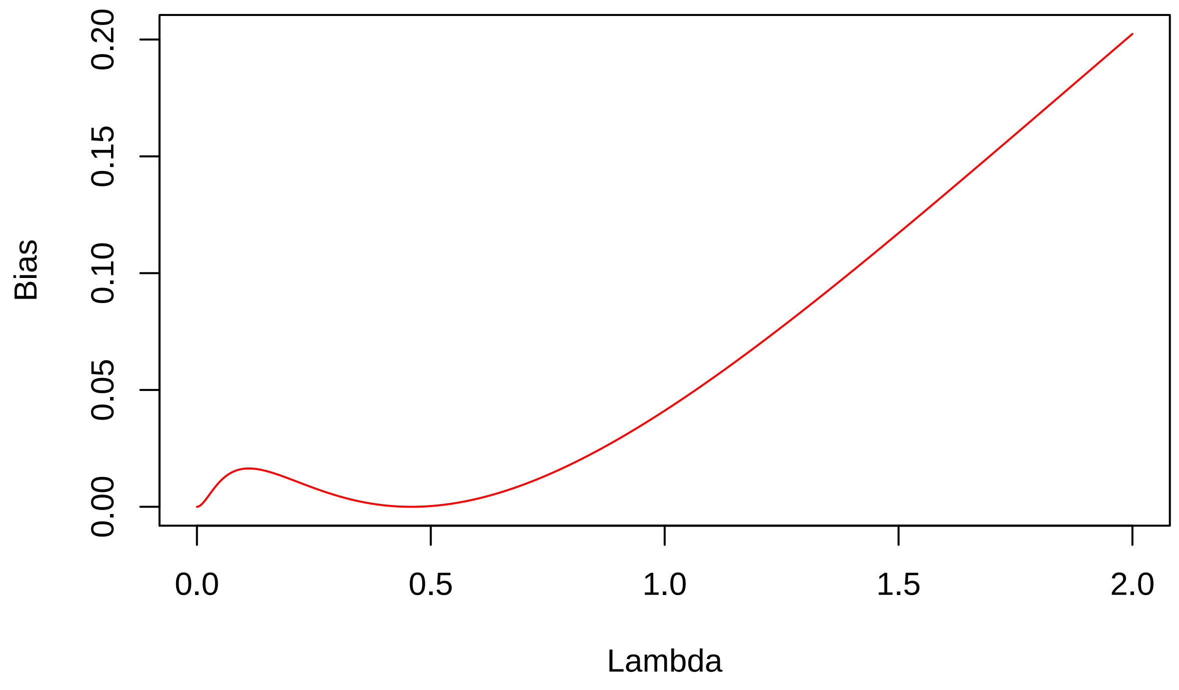
Hint: we perform matrix multiplication using `%*%`, transpose of a matrix `A` with `t(A)` and inverse with `solve(A)`.

Q22:

```r
sqbias=function(lambda,X,x0,beta)
{
  p=dim(X)[2]
  W = solve(t(X) %*% X + lambda*diag(p)) %*% t(X)
  value= (t(x0)%*%(W%*%X - diag(p))%*%beta)^2
  return(value)
}


thislambda=seq(0,2,length=500)
```

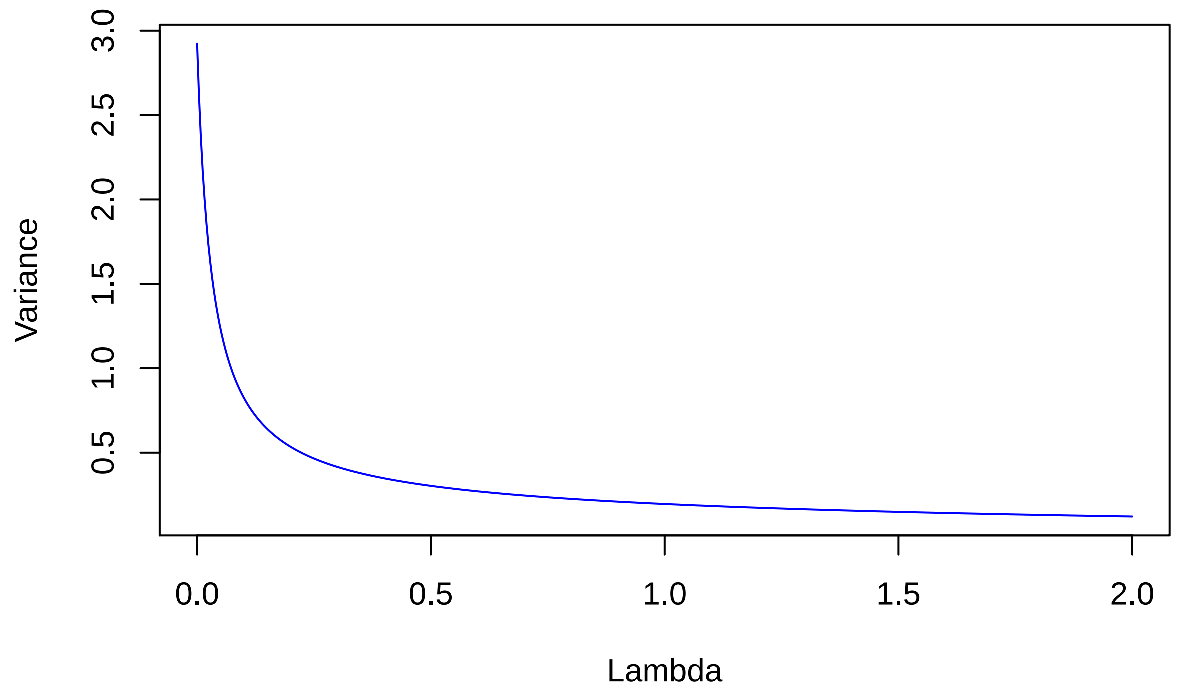
```
sqbiaslambda=rep(NA,length(thislambda))
for (i in 1:length(thislambda)) sqbiaslambda[i]=sqbias(thislambda[i],X,x0,beta)
plot(thislambda,sqbiaslambda,col=2,type="l",ylab="Bias",xlab="Lambda")
```



The curve looks somewhat like what I expected. I expected the squared bias to be 0 for $\lambda = 0$, as then the ridge regression estimator is equal to the least square estimator, which is unbiased. I then expected the bias to increase monotonically for increasing values of λ as the constraints on the estimator coefficients would become stricter and stricter. However from the plot one can see that the bias does not increase monotonically, but rather has a local minima around $\lambda = 0.5$.

Q23:

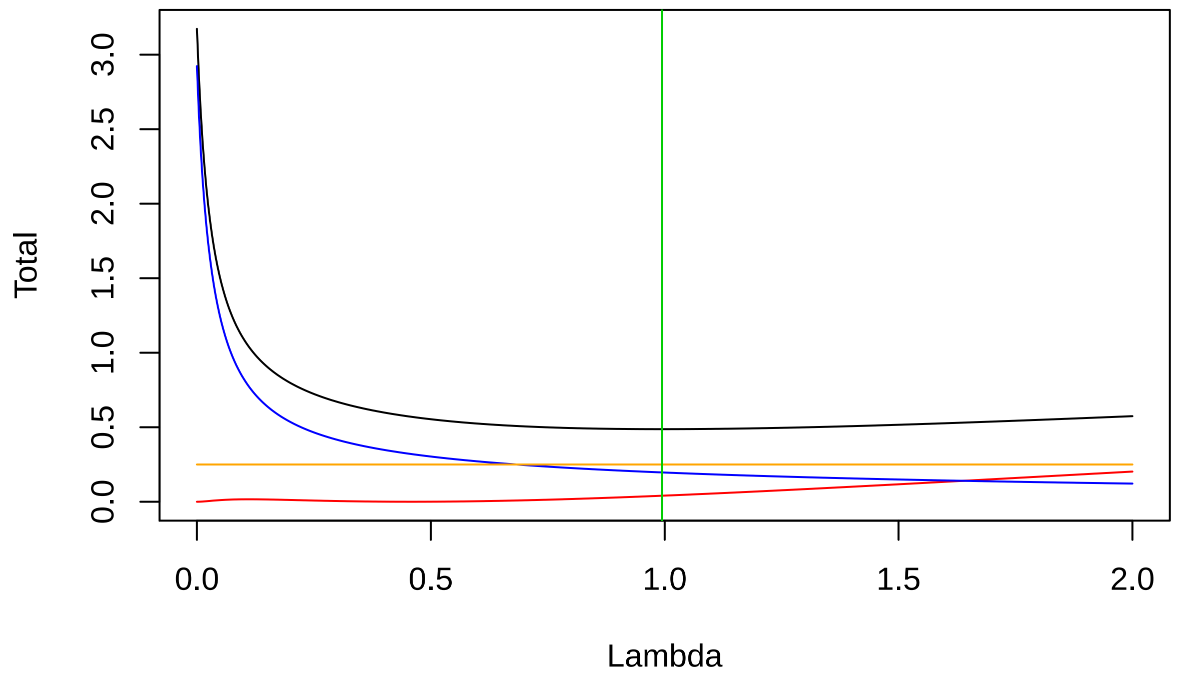
```
variance=function(lambda,X,x0,sigma)
{
  p=dim(X)[2]
  W = solve(t(X) %*% X + lambda*diag(p)) %*% t(X)
  value = sigma^2 * t(x0)%*%W%*%t(W)%*%x0
  return(value)
}
thislambda=seq(0,2,length=500)
variancelambda=rep(NA,length(thislambda))
for (i in 1:length(thislambda)) variancelambda[i]=variance(thislambda[i],X,x0,sigma)
plot(thislambda,variancelambda,col=4,type="l",ylab="Variance",xlab="Lambda")
```



The curve for the Var$[\widetilde{f}(\lambda)]$ does look like expected. From the expression of Var$[\widetilde{f}(\lambda)]$ one can see that there are two factors including λ, both of which are inversed. In the case that $\mathbf{X}^T\mathbf{X}$ is positive, all the terms in the expression for the variance are positive. This means that the expression for the variance is strictly decreasing for increasing values of λ. This makes sense when one thinks about λ as a constraint on the estimator coefficients. More constraints means smaller coefficient values and hence less variance in the model.

Q24:

```r
tot=sqbiaslambda+variancelambda+sigma^2
lambda_optimal = thislambda[which.min(tot)]
plot(thislambda,tot,col=1,type="l",ylim=c(0,max(tot)),ylab="Total",xlab="Lambda")
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)
```



The optimal value for λ in this case is 0.994.