Compulsory exercise 1: Group 13

TMA4268 Statistical Learning V2019

Øyvind Klåpbakken, Martin Outzen Berild and Sindre Henriksen

February 22nd 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: The equation for the fitted 'modelA' is

$$\log(FEV) = -1.944 + +.0234 \times \mathit{Age} + 0.0168 \times \mathit{Htcm} + 0.0293 \times \mathit{GenderM} - 0.0460 \times \mathit{Smoke}\,,$$

where `GenderM` is 0 if gender is female and 1 if gender is male. The true model is assumed to be

$$\log(FEV) = \beta_0 + \beta_1 \times \mathit{Age} + \beta_2 \times \mathit{Htcm} + \beta_3 \times \mathit{GenderM} + \beta_4 \times \mathit{Smoke} + \epsilon\,,$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

Q2:

- `Estimate` values are the estimates of the coefficients $\hat{\beta} = (X^\intercal X)^{-1} X^\intercal Y$. The intercept estimate is the value of `log(FEV)` if all the covariates are 0. For the covavariates the `Estimate` values say how much `log(FEV)` increases with one covariate if all the other covariates are kept constant (according to the model). The model point estimate of `log(FEV)` for given covariate values is found by using the formula in Q1. So if e.g. `Age` is increased by 1 and all the other covariates are kept constant, the estimate of `log(FEV)` increases by 0.0234.

- `Std.Error` is the standard error of the estimated coefficients (the values under `Estimate`). This tells how much the estimates will vary if we repeat the experiment with the same covariate values if our model assumptions are true. The values are the diagonal elements of the coefficient estimate covariance matrix $Cov(\hat{\beta}) = (X^\intercal X)^{-1}\sigma^2$.
- `Residual standard error` is an unbiased estimator for the standard deviation σ of the error term ϵ in our model $y = \beta^\intercal x + \epsilon$, where β is the vector of coefficients and $x_i = (1, Age_i, Htcm_i, GenderM_i, Smoke_i)$. It is given by

$$RSE = \hat{\sigma} = \frac{RSS}{n-p-1}\,, \quad \text{where} \quad RSS = \sum_{i=1}^{n}(Y_i - \hat{Y})^2$$

  and n is the number of observations and p the number of covariats (4).
- `F-statistic` is a statistic for the null hypothesis $H_0 : \beta_1 = \beta_2 = \beta_3 = \beta_4 = 0$ vs the alternative hypothesis H_1: not all the coefficients (except the intercept) are 0. The p-value is very small, so in this case H_0 is clearly rejected, i.e. the regression is significant. The value and the p-value is given by

$$F = \frac{(TSS - RSS)/p}{RSS/(n-p-1)} \sim F_{p,n-p-1}\,, \quad TSS = \sum_{i=1}^{n}(Y_i - \bar{Y})^2\,.$$

  p and $n-p-1$ are the degrees of freedom of the F-distribution.

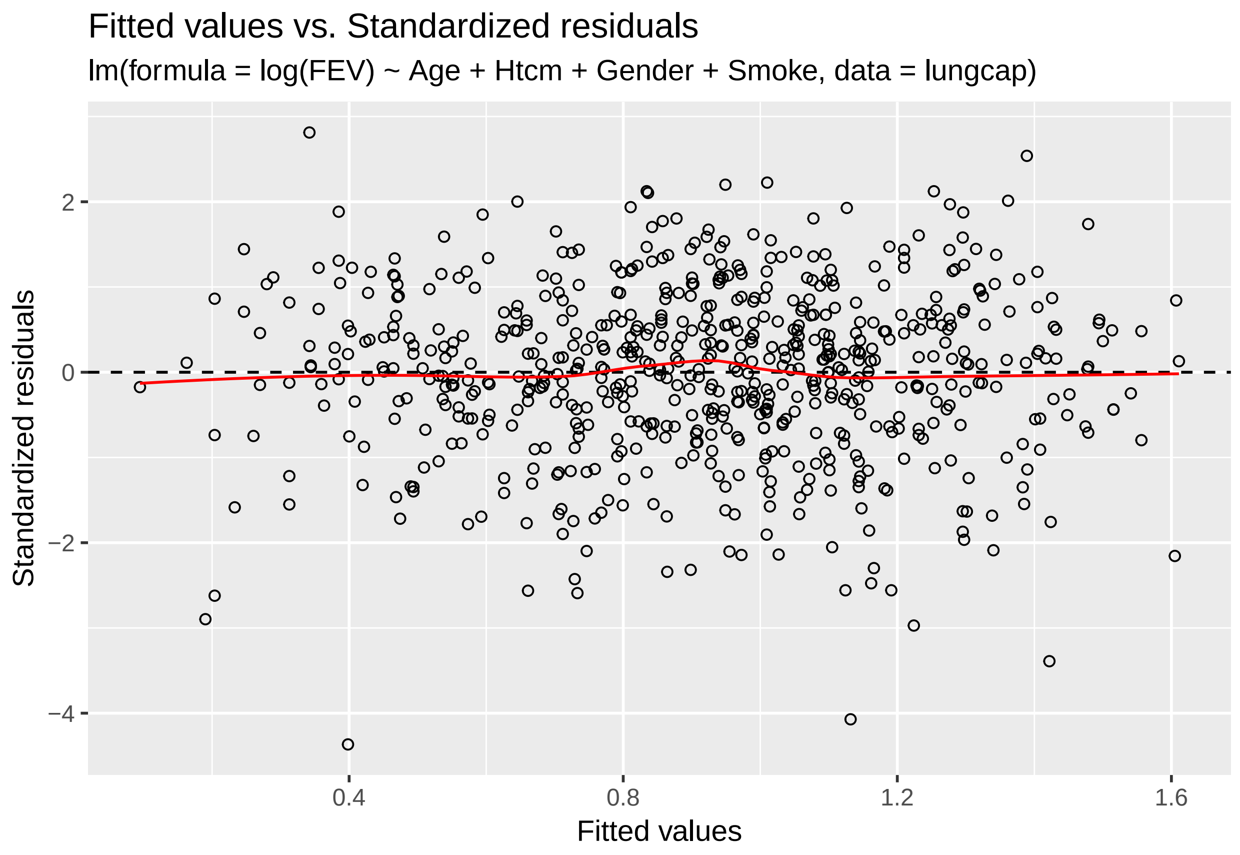
Q3: Multiple R-squared is the proportion of the variability in the response variable explained by the model, i.e.

$$\frac{TSS - RSS}{TSS}\,.$$

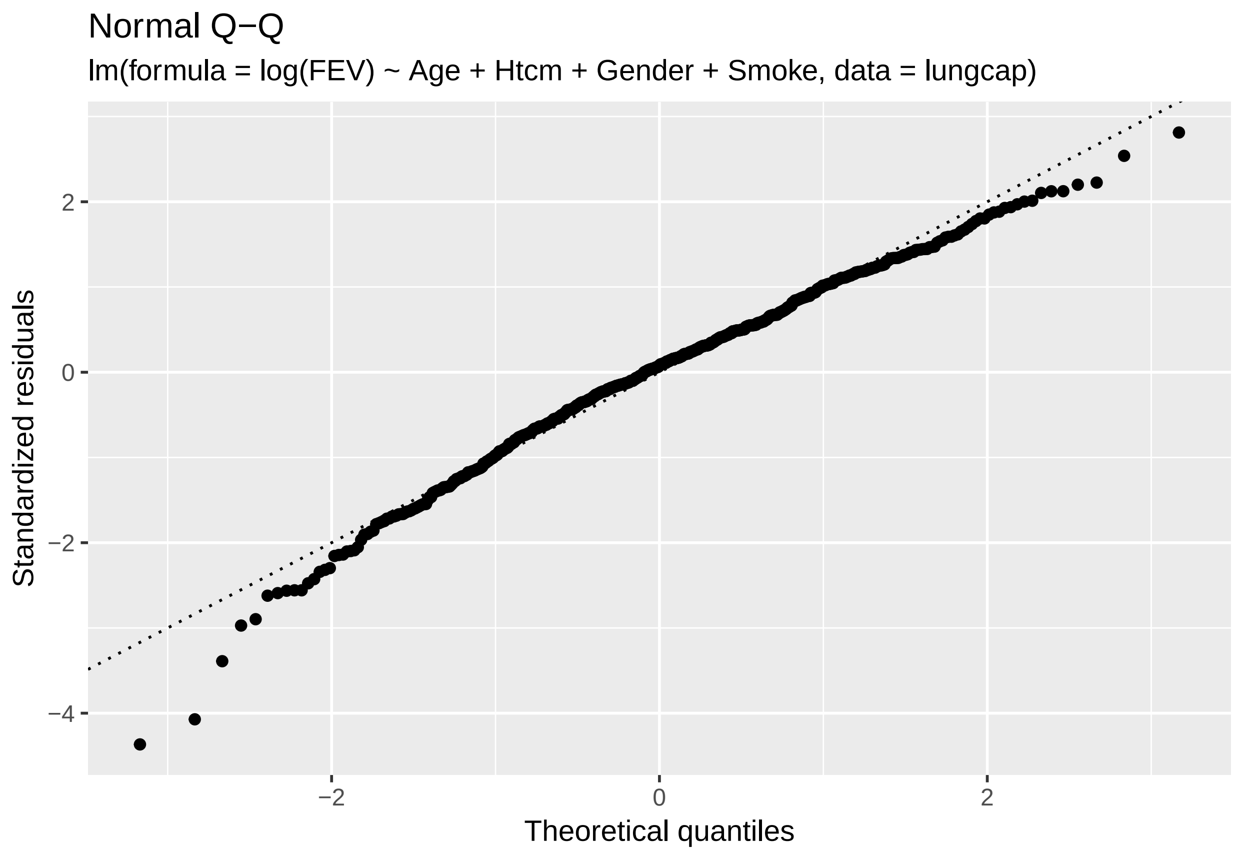
So here 81.06% of the variability is explained by the model.

Q4:

```r
library(ggplot2)
# residuals 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))
```

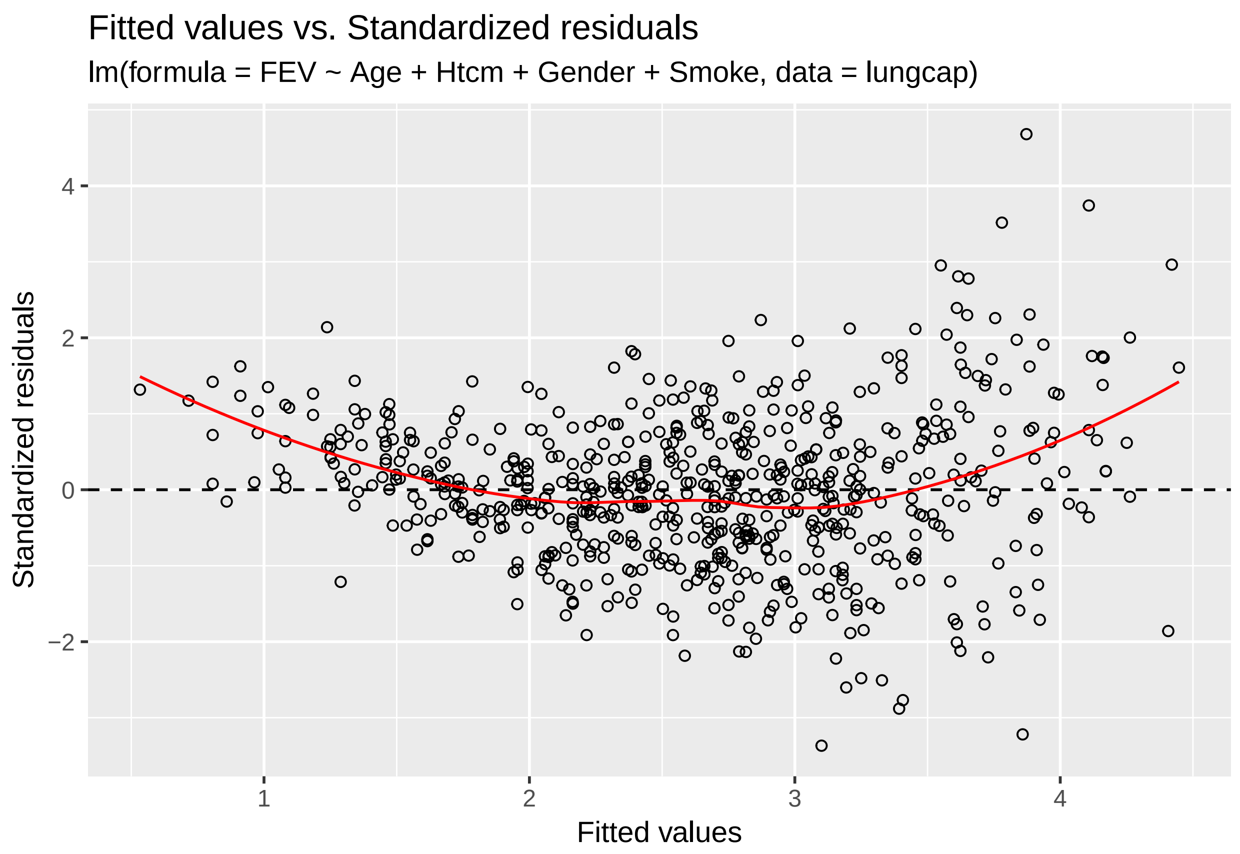
```
##
##  Anderson-Darling normality test
##
## data:  rstudent(modelA)
## A = 1.9256, p-value = 6.486e-05
```

The plot of standardized residuals shows that there is no clear trend, i.e. the residuals seem to be randomly distributed around 0 for all fitted values. We also can't see any covariance structure from this plot. So this plot does not indicate that any of our model assumptions are incorrect.

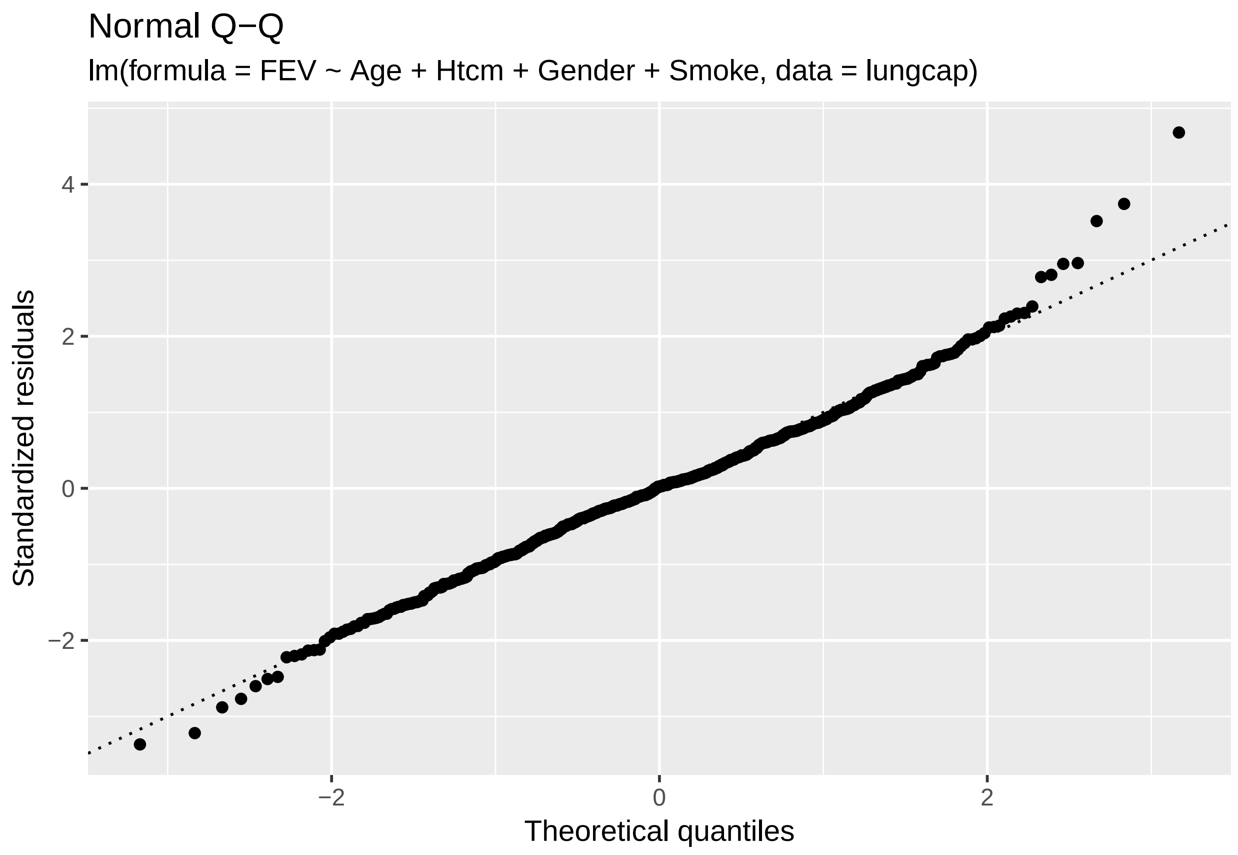
The normal QQ-plot, however, shows that the standardized residuals are not perfectly normally distributed. This is confirmed by the low p-value from the Anderson-Darling normality test, leading to rejection of the null hypothesis that the residuals are normally distributed.

Q5:

```r
modelB = lm(FEV ~ Age + Htcm + Gender + Smoke, data=lungcap)
summary(modelB)

# residuals 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))
```



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

```
##
## 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
##
##
```

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

The model with `FEV` as response instead of `log(FEV)` is fitted an testet in the code above. We see that the proportion of variability explained is somewhat smaller than before (77.54%) (we don't need to look at adjusted R-squared since the number of covariates in the models are the same). Also, the standardized residuals do not seem to be distributed around 0 for all fitted values. The studentized residuals are somewhat more likely to be normally distributed, but still get a p-value as low as 0.4% from the Anderson-Darling test. Notable is the fact that `Smoke` is no longer considered to have a significant explanatory power, which makes room for a more parsimonous model. But then again, smoking probably does affect lung capacity. All in all `modelA` is preferable, since our model assumptions seem more reasonable for `log(FEV)` as response.

Q6: We wish to test $H_0 : \beta_{Age} = 0$ against $H_1 : \beta_{Age} \neq 0$. We use the test statistic

$$T_{Age} = \frac{\hat{\beta}_{Age}}{\sqrt{c_{jj}}\hat{\sigma}} \sim t_{n-p-1} \,,$$

where c_{jj} is the second diagonal element of $X^\intercal X)^{-1}$. The calculations are already performed by `lm` and printed in the summary before. The t-value is 6.984, resulting in a p-value of 7.1e-12, so the null hypothesis is rejected at any reasonable signifiance level, i.e. `Age` has a significant effect on the response. Common siginficance levels are 0.05 and 0.01.

Q7:

```r
beta_age = modelA$coefficients[2][["Age"]]
alpha = 0.01
p = 4
n = nrow(lungcap)
t = qt(alpha/2, n-p-1)
se_beta_age = sqrt(vcov(modelA)[2,2])
CI99 = c(beta_age + t*se_beta_age, beta_age - t*se_beta_age)
CI99
```

```
## [1] 0.01473674 0.03203769
```

A 99% confidence interval for β_{Age} is found and printed above, using

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

where $\sqrt{c_{jj}}\hat{\sigma} = SE(bet\hat{a}_{Age})$. This interval tells us that when repeating the experiment and measuring the response over and over, 99% of such intervals will contain the true value β_{age}. Since the interval does not contain the value 0, we can say for sure that the p-value for the test in Q6 is less than 1%.

Q8:

```r
new = data.frame(Age=16, Htcm=170, Gender="M", Smoke=0)
log_prediction = predict.lm(modelA, new, interval="prediction", level=0.95)
log_prediction
prediction = exp(log_prediction)
prediction
```

```
##        fit     lwr      upr
## 1 1.323802 1.03616 1.611444
##       fit      lwr      upr
## 1 3.75768 2.818373 5.010038
```

Our best guess for the value of `log(FEV)` for new person is 1.324. A 95% prediction interval for his `FEV` is (2.818, 5.010). This tells us that we can say with 95% certainty (given that our model assumptions are true) that he has a relatively high lung capacity; possibly close to the sample maximum 5.793 and likely not close to the minimum 0.791. The interval is, however, quite big.

Problem 2: Classification

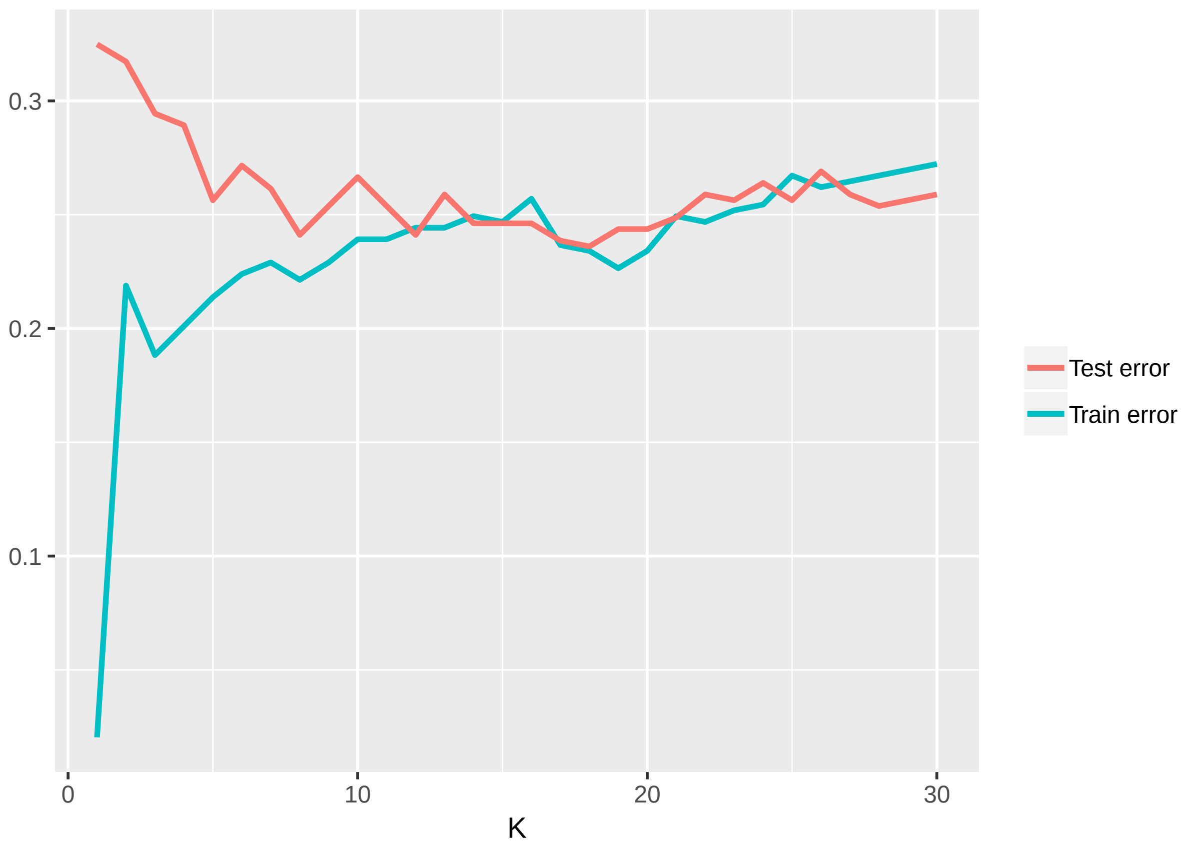
```r
library(class)# for function knn
library(lattice)
library(caret)# for confusion matrices
library(ggplot2)

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 KNN estimator $\hat{y} \in 0, 1$ is determined by taking a "majority vote" of the N closest neighbours of x.

$$\hat{y} = \hat{P}(Y = j|X = x_0) = \frac{1}{K} \sum_{i \in \mathcal{N}_0} I(y_i = j) = \begin{cases} \frac{1}{K} \sum_{i \in N} y_i, & j = 1 \\ 1 - \frac{1}{K} + \sum_{i \in N} y_i, & j = 0 \end{cases} \tag{1}$$

Q10: Finding the the test and training error of the data set

```r
ks = 1:30
train.var = Mdf[Mdf$istest == 0, 2:3]
test.var = Mdf[Mdf$istest == 1, 2:3]
train.def = Mdf[Mdf$istest == 0, 1]
test.def = Mdf[Mdf$istest == 1, 1]
train.e = numeric(30)
test.e = numeric(30)
for (k in ks){
  model1 <- class::knn(train = train.var, test = test.var, cl = train.def, k = k, prob=TRUE)
  model2 <- class::knn(train = train.var, test = train.var, cl = train.def, k = k, prob=TRUE)
  train.e[k] = mean(train.def != model2)
  test.e[k] = mean(test.def != model1)
}
ggplot(data.frame(K=1:30,train.e,test.e),aes(x=K)) +
  geom_line(aes(y = train.e, col = "Train error"),size = 1) +
  geom_line(aes(y = test.e, col = "Test error"),size = 1) +
  ylab("")+
  theme(legend.title = element_blank())
```



In the Figure above the training error and test error is displayed.

```r
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)
  })
})
```

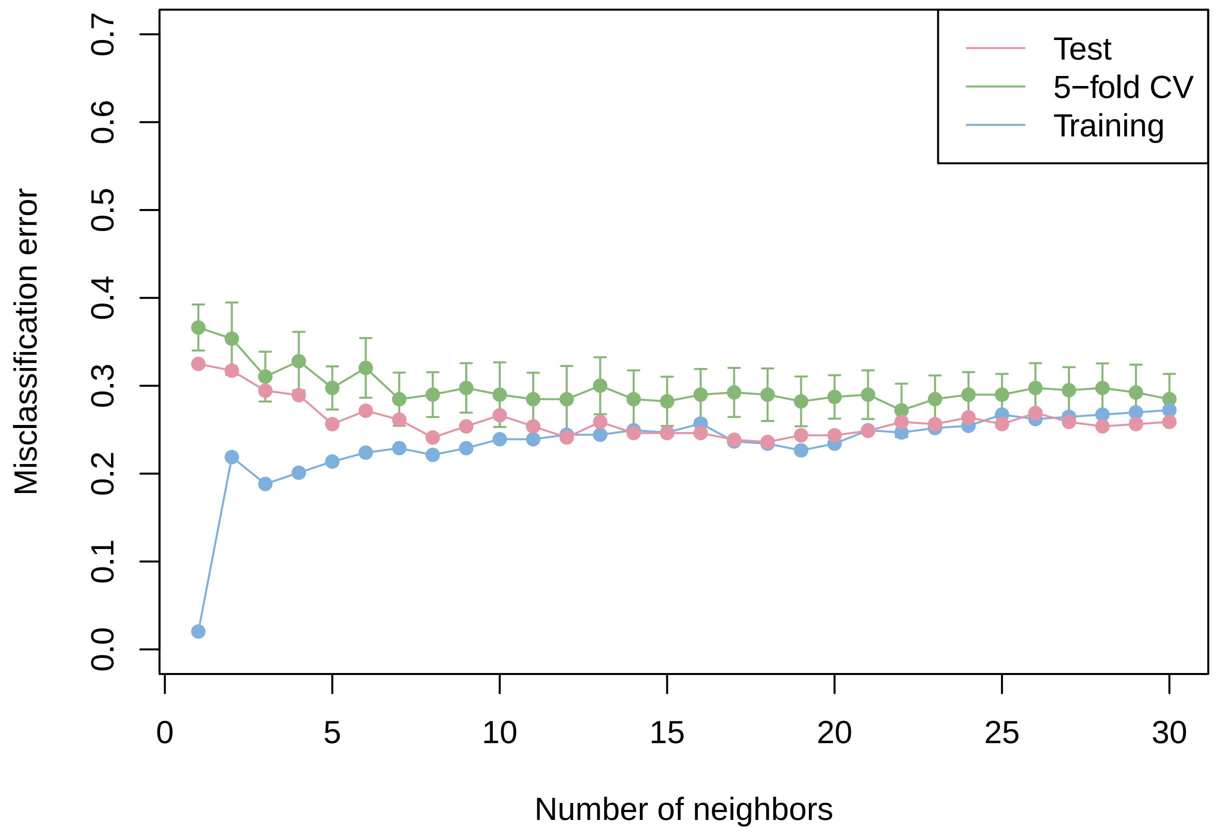
Q11: Calculating the average CV error, the standard error of the average CV error over the 5 folds and the $K$ corresponding to the smallest CV error.

```r
cv.e = colMeans(cv)
cv.se = apply(cv,2,sd)/sqrt(5)
k.min = which.min(cv.e)
cat("k.min = ", k.min)
```

```
## k.min =  22
```

Q12:

```r
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 for $\hat{y}(x)$ will increase with increasing K. If K is small there want be alot of bias, because the predictors are close to the point we are trying to estimate. The variance on the other hand will decrease with increasing K. If K is small the random noise will affect it the estimate more than if there were more points. You can also see this from the formula of variance.

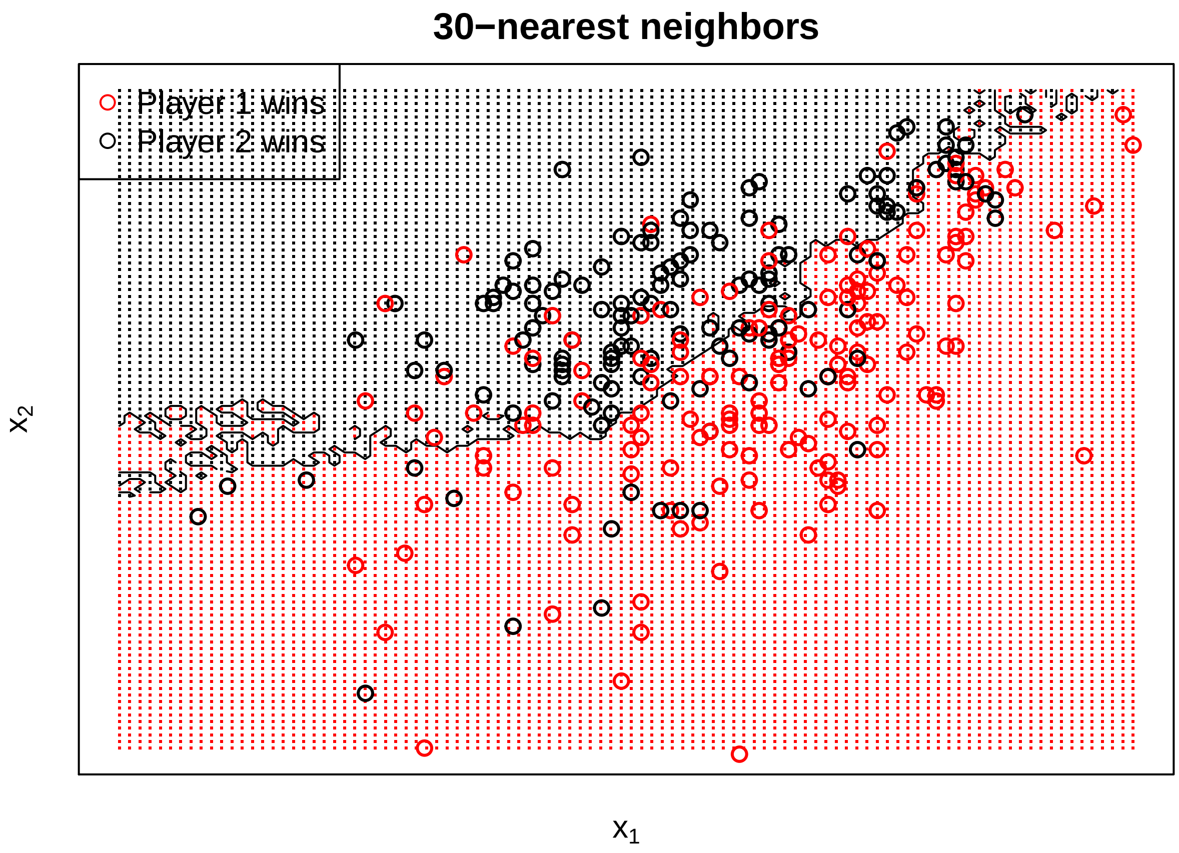
Q13: The strategy used in the first line is the *one standard error rule*, which means that we choose the highest K for which or

$$K = \max_K \{CV(K) \leq CV(\hat{K}) + SE(\hat{K})\},$$

where $\hat{K}$ is the k.min from Q11.

```r
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)
```

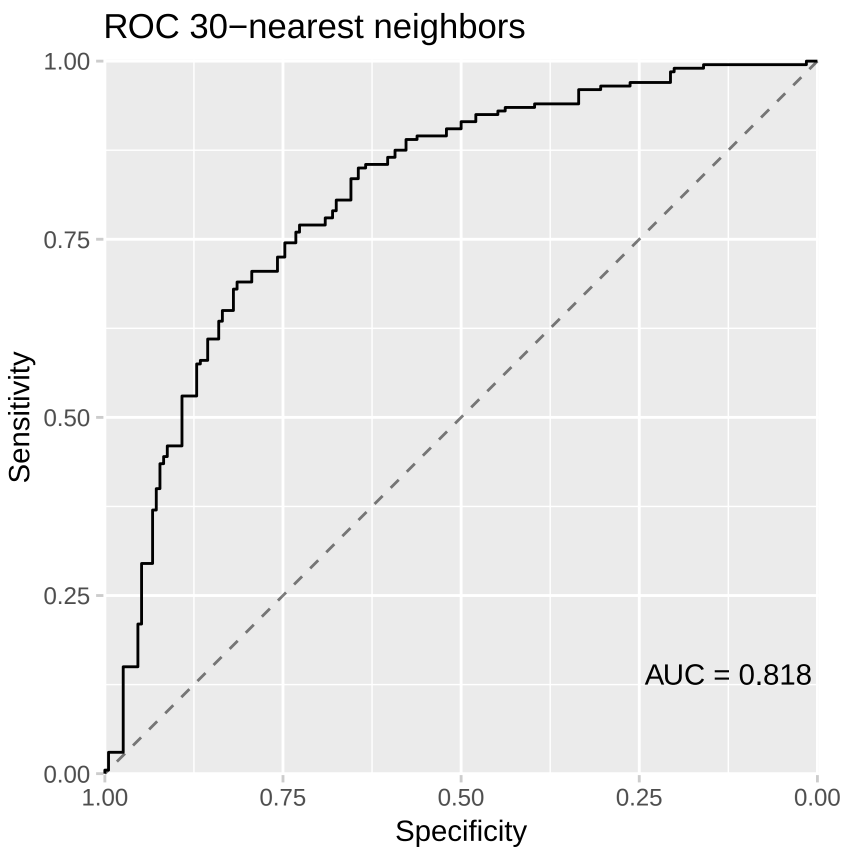
```
np = 300
par(mar=rep(2,4), mgp = c(1, 1, 0))
contour(xnew[,1], xnew[,2], z = matrix(grid.yhat, size), levels=.5,
        xlab=expression("x"[1]), ylab=expression("x"[2]), axes=FALSE,
        main = paste0(k,"-nearest neighbors"), cex=1.2, labels="")
points(grid, pch=".", cex=1, col=grid.yhat)
points(M[1:np,-1], 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()
```



Q14: The optimal choice is $K = 30$ from Q13.

```
K=tail(which(cv.e < cv.e[k.min] + cv.se[k.min]), 1)

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)

library(pROC)
myRoc <- roc(response = M[-tr,1], predictor = KNNprob, auc = TRUE, ci = TRUE)
interval = 0.25
breaks = seq(0,1,interval)
ggplot(NULL, aes(x =rev(myRoc$specificities), y =rev(myRoc$sensitivities)))+
  geom_segment(aes(x = 0, y = 1, xend = 1,yend = 0), linetype = "dashed", alpha = 0.5) +
  geom_step() +
  scale_x_reverse(name = "Specificity",limits = c(1,0), breaks = breaks, expand = c(0.001,0.001)) +
```

```r
  scale_y_continuous(name = "Sensitivity", limits = c(0,1), breaks = breaks, expand = c(0.001, 0.001))
  theme(axis.ticks = element_line(color = "grey80")) +
  coord_equal() +
  annotate("text", x = interval/2, y = interval/2, vjust = 0, label = paste("AUC =",sprintf("%.3f",myRo
  ggtitle("ROC 30-nearest neighbors")
```



The ROC curve displays the relationship between the percentage of correctly predicted negatives of true negatives, *specificity*, and the percentage of correcly predicted positives of true positives, *sensitivity*. Their respective equations are

$$\text{specificity} = \frac{\#\text{true negatives}}{\#\text{negatives}} = \frac{TN}{N},$$

$$\text{sensitivity} = \frac{\#\text{true positives}}{\#\text{positives}} = \frac{TP}{P}.$$

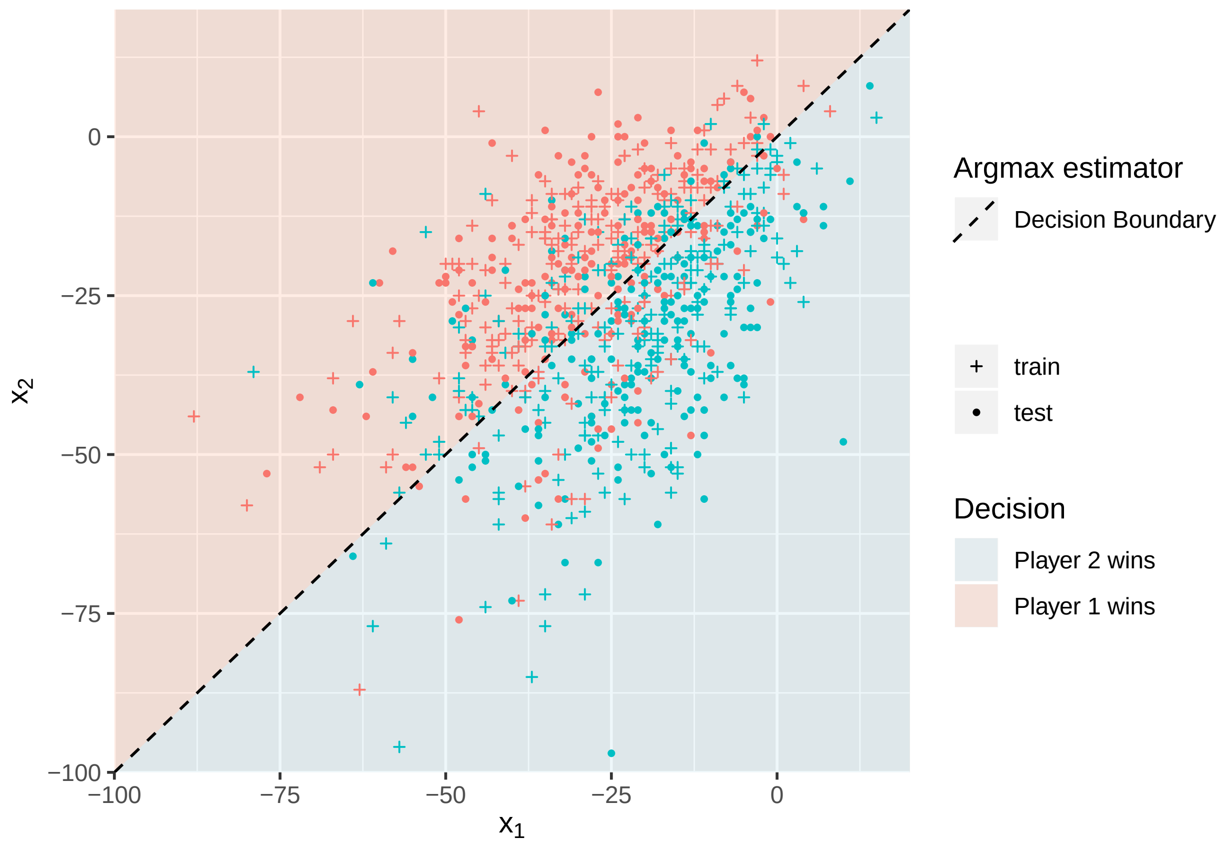
The x-axis being the specificity is flipped so that specificity $= 1$ is to the left in the ROC. The best scenario is if we don't have any missclassifications, then the ROC curve would be in the top left corner, and the *"area under the curve"*(AUC) would be 1. We had AUC $= 0.8178093$.

If we have a predictor that classify a fraction p as positive, then p percent of the predictions would be true positives. The same predictor would classify a fraction p as negative, the p percent of the predictions would be true negative. Therefore true positive rate is equal to true negative rate, and we have a line from (1,0) to (0,1) as the ROC curve. This would yield a AUC $= 0.5$.

Q15:

Now we use a different predictor, $\tilde{y} = \text{argmax}_k(x_k)$

```r
  quality_score <- data.frame(Mdf, y_hat = as.integer(M$x1>M$x2))
  whoWins <- data.frame(
    x1=c(-100,20,20,-100,20,-100),
    x2=c(-100,-100,20,-100,20,20),
    win=c(1,1,1,2,2,2)
  )
ggplot(quality_score)+
    geom_polygon(data = whoWins, aes(x=x1,y=x2,group=win,fill=factor(win)),alpha=0.2)+
    scale_fill_manual(name = "Decision",labels=c("Player 2 wins","Player 1 wins",""),values = c("lightbl
    geom_point(aes(x = x1, y= x2, col = y,shape=istest),size =1)+
    scale_shape_manual(name ="", label=c("train","test"),values=c(3,16))+
    guides(color=FALSE)+
    geom_abline(aes(slope=1,intercept=0,linetype = "Decision boundary"))+
    xlab(expression("x"[1])) +
    ylab(expression("x"[2])) +
    scale_x_continuous(limits = c(-100, 20), expand = c(0, 0)) +
    scale_y_continuous(limits = c(-100, 20), expand = c(0, 0)) +
    scale_linetype_manual(name="Argmax estimator",labels="Decision Boundary",values="dashed")
```



```r
print("Argmax predictor")
cM_argmax <- confusionMatrix(table(quality_score$y_hat[-tr],quality_score$y[-tr]))
cM_argmax
print("KNN predictor")
cM_KNN <- confusionMatrix(table(KNNclass,quality_score$y[-tr]))
cM_KNN
```

```
## [1] "Argmax predictor"
## Confusion Matrix and Statistics
##
##
##       0   1
##   0 149  47
##   1  45 153
##
##                Accuracy : 0.7665
##                  95% CI : (0.7215, 0.8074)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.533
##  Mcnemar's Test P-Value : 0.917
##
##             Sensitivity : 0.7680
##             Specificity : 0.7650
##          Pos Pred Value : 0.7602
##          Neg Pred Value : 0.7727
##              Prevalence : 0.4924
##          Detection Rate : 0.3782
##    Detection Prevalence : 0.4975
##       Balanced Accuracy : 0.7665
##
##        'Positive' Class : 0
##
## [1] "KNN predictor"
## Confusion Matrix and Statistics
##
##
## KNNclass   0   1
##        0 137  45
##        1  57 155
##
##                Accuracy : 0.7411
##                  95% CI : (0.6949, 0.7837)
##     No Information Rate : 0.5076
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4816
##  Mcnemar's Test P-Value : 0.2761
##
##             Sensitivity : 0.7062
##             Specificity : 0.7750
##          Pos Pred Value : 0.7527
##          Neg Pred Value : 0.7311
##              Prevalence : 0.4924
##          Detection Rate : 0.3477
##    Detection Prevalence : 0.4619
##       Balanced Accuracy : 0.7406
##
##        'Positive' Class : 0
##
```

From from the confusion matrices we see that specificity is better for the KNN predictor, but the sensitivity is better for the argmax predictor. From the misclassficiation error for KNN, 0.2588832, and the argmax, 0.2335025, We see that the argmax predictor is the best one.

Problem 3: Bias-variance trade-off

Here you see how to write formulas with latex (needed below)

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

Q16:

Utilizing that $\mathrm{E}(Y) = X\beta$ and $\mathrm{E}(CX) = C\mathrm{E}(X)$ we obtain that $\hat{\beta}$ is an unbiased estimator for β.

$$\mathrm{E}(\hat{\beta}) = \mathrm{E}\left[(X^TX)^{-1}X^TY\right] = (X^TX)^{-1}X^T\mathrm{E}(Y) = (X^TX)^{-1}X^TX\beta = \beta$$

By using that $\mathrm{Cov}(CY) = C\mathrm{Cov}(Y)C^T$ we obtain

$$\mathrm{Cov}(\hat{\beta}) = \mathrm{Cov}\left[(X^TX)^{-1}X^TY\right] = (X^TX)^{-1}X^T\mathrm{Cov}(Y)\left[(X^TX)^{-1}X^T\right]^T = \sigma^2(X^TX)^{-1}X^TX(X^TX)^{-1} = \sigma^2(X^TX)^{-1}$$

Q17:

Using the same tools as above, we obtain

$$\mathrm{E}\left[\hat{f}(\mathbf{x_0^T})\right] = \mathrm{E}(\mathbf{x_0^T}\hat{\beta}) = \mathbf{x_0^T}\mathrm{E}(\hat{\beta}) = \mathbf{x_0^T}\beta$$

$$\mathrm{Var}\left[\hat{f}(\mathbf{x_0^T})\right] = \mathrm{Var}(\mathbf{x_0^T}\hat{\beta}) = \mathbf{x_0^T}\mathrm{Cov}(\hat{\beta})\mathbf{x_0} = \boldsymbol{\sigma^2}\mathbf{x_0^T}(\mathbf{X^TX})^{\mathbf{-1}}\mathbf{x_0}$$

Q18:

$$\mathrm{E}[(Y_0 - \hat{f}(\mathbf{x}_0))^2] = [\mathrm{E}(\hat{f}(\mathbf{x}_0) - f(\mathbf{x}_0)]^2 + \mathrm{Var}(\hat{f}(\mathbf{x}_0)) + \mathrm{Var}(\varepsilon)$$

We start off by noting that $\mathrm{E}(f(\mathbf{x_0})) = \mathrm{E}(\mathbf{Y_0} - \epsilon) = \mathrm{E}(\mathbf{x_0^T}\beta)$, which enables us to write $(\mathbf{x_0^T} - \mathbf{x_0^T}\hat{\beta})^{\mathbf{2}}$ as $\mathrm{E}(f(\mathbf{x_0}) - \hat{\mathbf{f}}(\mathbf{x_0}))^{\mathbf{2}}$. The expression for $\mathrm{E}\left[(Y_0 - \hat{f}(x_0))^2\right]$ becomes

$$\mathrm{E}\left[(Y_0 - \hat{f}(x_0))^2\right] = \mathrm{Var}(Y_0 - \hat{f}(x_0)) + \mathrm{E}(Y_0 - \hat{f}(\mathbf{x_0}))^{\mathbf{2}} = \mathrm{Var}(\mathbf{x_0^T}\beta + \epsilon - \mathbf{x_0^T}\hat{\beta}) + \left(\mathrm{E}\left[\mathbf{x_0^T}\beta + \epsilon - \mathbf{x_0^T}\hat{\beta}\right]\right)^{\mathbf{2}}$$

$$= \mathrm{Var}(\epsilon) + \mathrm{Var}(\hat{f}(\mathbf{x_0})) + \left(\mathrm{E}\left[\mathbf{f}(\mathbf{x_0}) - \hat{\mathbf{f}}(\mathbf{x_0})\right]\right)^{\mathbf{2}}$$

Ridge estimator:

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

Q19:

We start off by defining $G := X^TX$, and by rewriting $\hat{\beta}$ as $\hat{\beta} = G^{-1}X^TY$. Note that G is a symmetric matrix. We can now write $\tilde{\beta}$ as

$$\tilde{\beta} = (G - \lambda I)^{-1}X^TY = (G - \lambda I)^{-1}GG^{-1}X^TY = (G + \lambda I)^{-1}G\hat{\beta}$$

$$= \left[G(I + \lambda G^{-1})\right]^{-1}G\hat{\beta} = (I + \lambda G^{-1})^{-1}G^{-1}G\hat{\beta} = (I + \lambda G^{-1})^{-1}\hat{\beta}$$

By defining $H := (I + \lambda G^{-1})$, we end up with

$$\tilde{\beta} = H^{-1}\hat{\beta}$$

It follows from the previous results, as well as the properties introduced earlier that

$$\mathrm{E}(\tilde{\beta}) = H^{-1}\mathrm{E}(\hat{\beta}) = H^{-1}\beta$$

and

$$\mathrm{Cov}(\tilde{\beta}) = H^{-1}\mathrm{Cov}(\hat{\beta})(H^{-1})^T = \sigma^2(HGH)^{-1}$$

Q20:

$$\mathrm{E}(\mathbf{x_0^T}\tilde{\beta}) = \mathbf{x_0^T}\mathbf{H^{-1}}\beta$$
$$\mathrm{Var}(\mathbf{x_0^T}\tilde{\beta}) = \sigma^{\mathbf{2}}\mathbf{x_0^T}(\mathbf{HGH})^{\mathbf{-1}}\mathbf{x_0}$$

Q21:

$$\mathrm{E}[(Y_0 - \widetilde{f}(\mathbf{x}_0))^2] = [\mathrm{E}(\widetilde{f}(\mathbf{x}_0) - f(\mathbf{x}_0)]^2 + \mathrm{Var}(\widetilde{f}(\mathbf{x}_0)) + \mathrm{Var}(\varepsilon)$$

Using that $\mathrm{Var}(X) = \mathrm{E}(X^2) - \mathrm{E}(X)^2$, we obtain

$$\mathrm{E}[(Y_0 - \tilde{f}(\mathbf{x_0}))^{\mathbf{2}}] = \mathrm{Var}(\mathbf{x_0^T}\beta + \epsilon - \mathbf{x_0^T}\tilde{\beta}) + \left(\mathrm{E}\left[\mathbf{x_0^T}\beta + \epsilon - \mathbf{x_0^T}\tilde{\beta}\right]\right)^{\mathbf{2}}$$
$$= \mathrm{Var}(\epsilon) + \mathrm{Var}(\tilde{f}(\mathbf{x_0})) + \left[\mathrm{E}(\mathbf{f}(\mathbf{x_0}) - \tilde{\mathbf{f}}(\mathbf{x_0})\right]^{\mathbf{2}}$$

```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
```

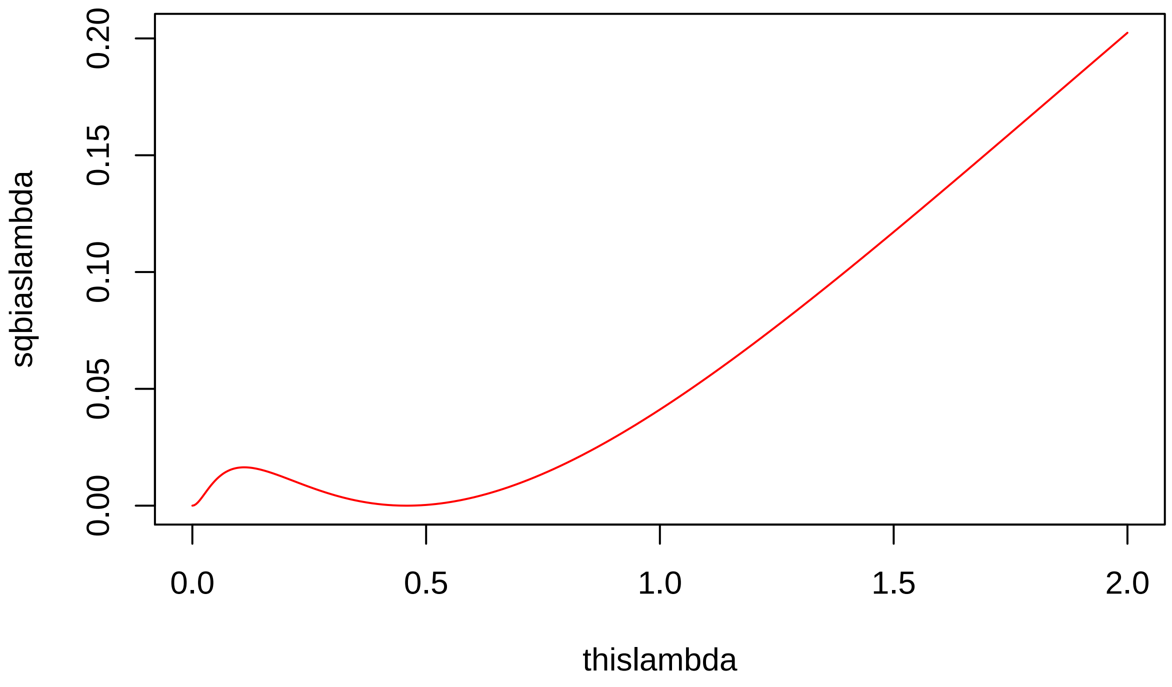
```
## [1] 100  81
## [1] 81  1
## [1] 81  1
## [1] 0.5
```

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

Q22:

The squared bias is zero for $\lambda = 0$, as one would expect from the results above. It was not expected to see that the squared bias is approaching zero again around $\lambda = 0.5$. Apart from this it develops as one would expect by increasing as *lambda* increases.

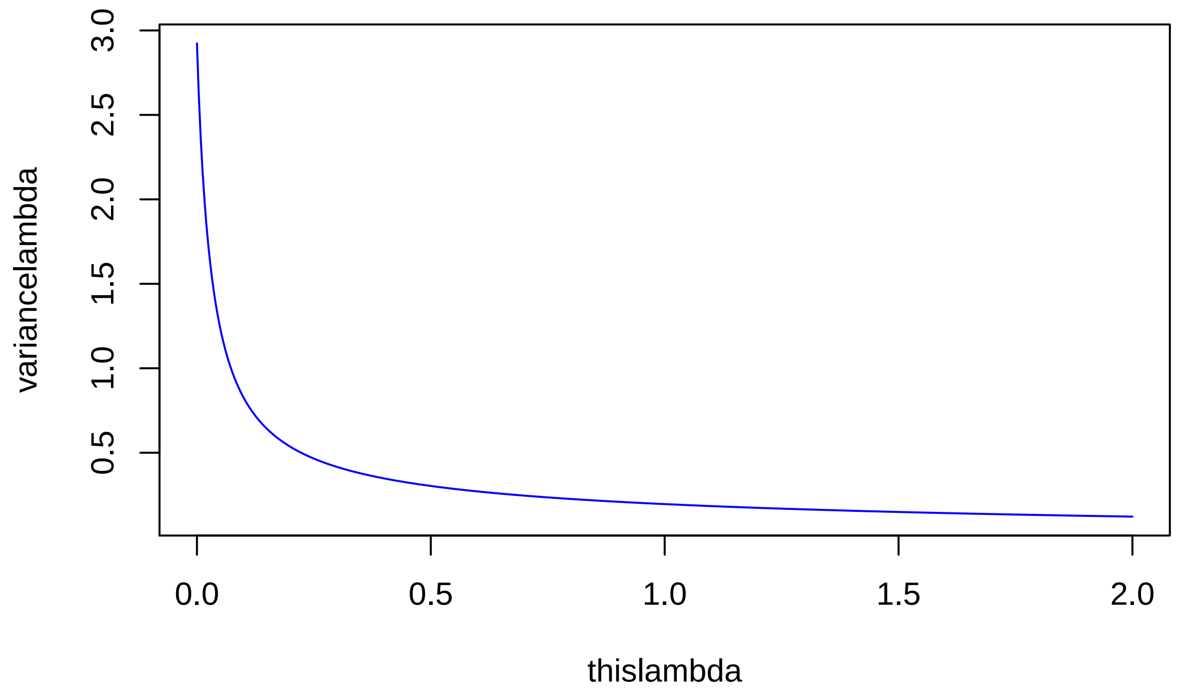
```r
sqbias=function(lambda,X,x0,beta)
{
  p=dim(X)[2]
  G <- t(X)%*%X
  H <- (diag(p) + lambda*solve(G))
  value <- (t(x0)%*%beta - t(x0)%*%solve(H)%*%beta)^2
  return(value)
}

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")
```



Q23:

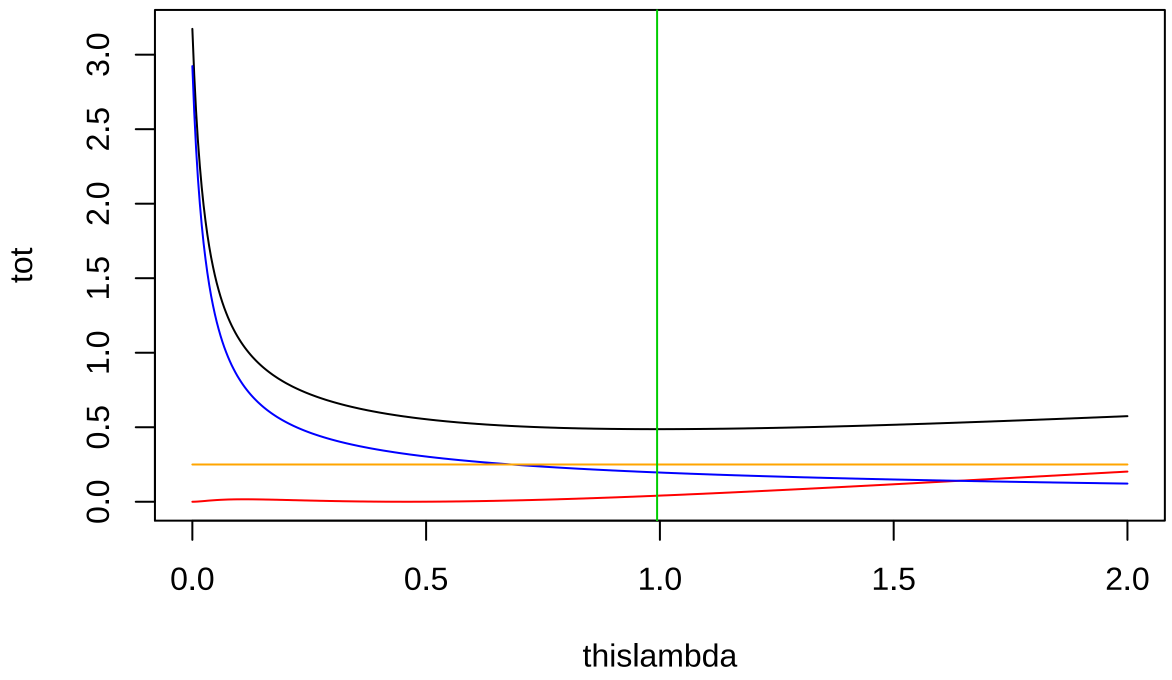
Having seen the previous plot, and by remembering that the expected squared prediction error is made up of the sum of the variance and the squared bias we would expect to see that the variance drops as the squared bias increases. Therefore, the plot of the variance is not surprising.

```r
variance=function(lambda,X,x0,sigma)
{
  p=dim(X)[2]
  inv=solve(t(X)%*%X+lambda*diag(p))
  G <- t(X)%*%X
  H <- (diag(p) + lambda*solve(G))
  value <- sigma^2*t(x0)%*%solve(H%*%G%*%H)%*%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")
```



Q24:

```
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)
```

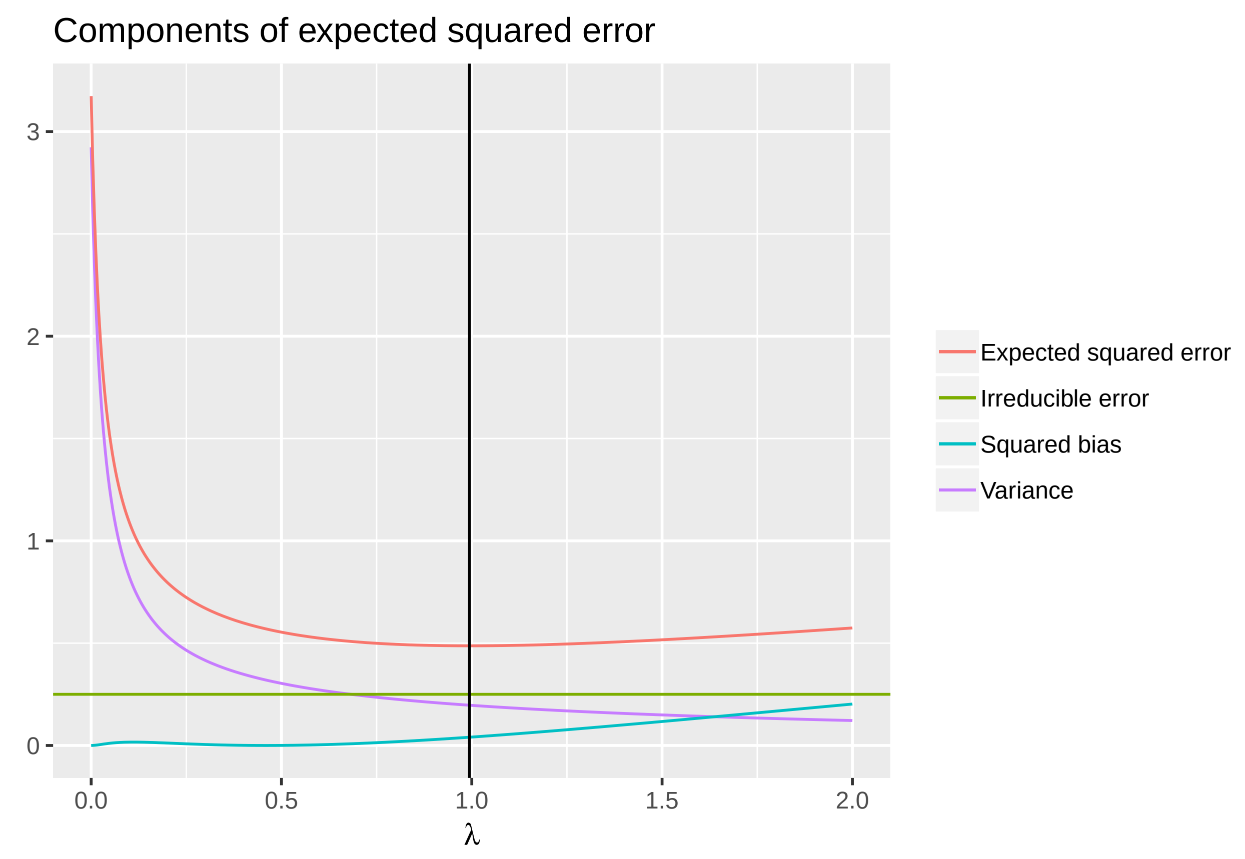


```
## [1] 249
## [1] 0.993988
```

The optimal value for λ is shown below. An alternative approach to plotting and finding this optimal value is also presented below. It serves the same purpose as what is done above.

```r
library(tidyverse)
```

```r
lambdas <- seq(0, 2, length = 500)
df <- tibble(lambda = lambdas,
             sqbias = lambdas %>%
               map_dbl(partial(sqbias, x0 = x0, beta = beta, X = X)),
             variance = lambdas %>%
               map_dbl(partial(variance, x0 = x0, sigma = sigma, X = X)))
df <- df %>% mutate(tot = sqbias + variance + sigma^2)
min <- df %>% filter(tot == min(tot)) %>% select(lambda, tot)
df %>% ggplot(aes(x = lambda)) +
  geom_line(aes(y = variance, color = 'Variance')) +
  geom_line(aes(y = sqbias, color = 'Squared bias')) +
  geom_line(aes(y = tot, color = 'Expected squared error')) +
  geom_hline(aes(color = 'Irreducible error', yintercept = sigma^2)) +
  geom_vline(aes(xintercept = min %>% select(lambda) %>% as.numeric())) +
  xlab(expression(lambda)) +
  ylab(NULL) +
  ggtitle('Components of expected squared error') +
  theme(legend.title = element_blank())
```



```
min %>% knitr::kable(caption = 'Optimal expected squared error',
              col.names = c('$\\lambda$', 'Expected squared error'),
              escape=FALSE)
```

Table 1: Optimal expected squared error

λ	Expected squared error
0.993988	0.4870826