Compulsory exercise 1: Group 64

TMA4268 Statistical Learning V2019

Jakob Gerhard Martinussen, Alm Wilson

19 February, 2019

Problem 1: Multiple linear regression

```
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 underlying distributional assumption is

$$\log(\text{FEV}) = Y = f(x) + \epsilon = x^T\beta + \epsilon = \beta_0 + \beta_{\text{age}}x_{\text{age}} + \beta_{\text{height}}x_{\text{height}} + \beta_{\text{male}}x_{\text{male}} + \beta_{\text{smoke}}x_{\text{smoke}} + \epsilon$$

Where we assume $\epsilon \sim N_n(0, \sigma^2 I)$. The *fitted* model is $\hat{Y} = x^T\hat{\beta}$.

Q2:

`Estimate`

The `Estimate` column represents the entries in the $\hat{\beta}$ vector of length 5, the model *estimation* for β. Remember that the model is fitted with a *logarithmic* FEV response, so the model prediction of an induvidual x's FEV becomes

$$\text{FEV} = e^{\hat{Y}} = e^{x^T\hat{\beta}}.$$

The `(Intercept)` estimate represents a non-smoking female individual of age 0 and height 0, which is nonsensical.

```
beta <- coefficients(modelA)
interceptFEV <- exp(beta["(Intercept)"])
interceptFEV
```

```
## (Intercept)
##   0.1431305
```

Such an imaginary individual is predicted to have a `FEV` of ≈ 0.14 litres. Now, in order to explain the remaining estimates, assume one of the covariates, x_j, to change to $x_j + 1$. How does the model's FEV prediction change from $\text{FEV}_{\text{before}}$ to $\text{FEV}_{\text{after}}$ in such a case?

$$\text{FEV}_{after} = e^{x^T\hat{\beta}} = e^{\hat{\beta}_0 + x_1\hat{\beta}_1 + \ldots + (x_j+1)\hat{\beta}_j + \ldots \hat{\beta}_p x_p} = e^{\hat{\beta}_j} e^{x^T\hat{\beta}} = e^{\hat{\beta}_j}\text{FEV}_{before}$$

The exponential of the coefficient estimates represents the *multiplicative change* of the model prediction as the respective covariate changes. In our case, these multiplicative effects are

```
library(tidyverse)
library(kableExtra)
enframe(exp(beta)) %>% kable()
```

name	value
(Intercept)	0.1431305
Age	1.0236628
Htcm	1.0169915
GenderM	1.0297534
Smoke	0.9549775

Here we see that an individual which smokes is expected to have $\approx 95.5\%$ of the `FEV` of a non-smoking individual, everything else being equal. Likewise, for every additional year of age, an individual is expected to increase their `FEV` by $\approx 2.4\%$.

`Std.Error`

The `Std.Error` column shows the model estimate of the standard error, $\text{SE}(\beta_j)$. In order to calculate these estimates, we need the notion of *residuals*, defined by

$$e_i := Y_i - \hat{Y}_i = Y_i - x_i^T\hat{\beta},$$

i.e. the difference between the actually observed values and the model predictions for the training set. The population distribution variance parameter can now be estimated by

$$\hat{\sigma}^2 = \frac{\sum_{i=1}^n e_i^2}{n-p-1} = \frac{\text{RSS}}{n-p-1},$$

where RSS is the residual sum of squares. The estimation of the standard error of the parameter estimation for β_j can now be expressed as

$$\hat{\text{SE}}(\hat{\beta}_i) = \sqrt{[(X^T X)^{-1}]_{[i,i]}}\ \hat{\sigma} := \sqrt{c_{jj}}\ \hat{\sigma},$$

where c_{jj} is the j'th diagonal entry of the matrix $(X^T X)^{-1}$. For instance, under our model assumptions, β_{age} is estimated to be normally distributed with standard error

```
modelASummary <- summary(modelA)
modelAEstimates <- as_tibble(modelASummary$coefficients)
modelAEstimates$`Std. Error`[2]
```

```
## [1] 0.003348451
```

Residual standard error

The residual standard error (RSS) for our model is

```
modelASummary$sigma
```

```
## [1] 0.1454686
```

and is found by summing the square difference between the observed values and the model predictions, as follows

$$\text{RSS} = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 = \sum_{i=1}^{n} (Y_i - x_i^T \hat{\beta})^2 = (Y - X\hat{\beta})^T (Y - X\hat{\beta})$$

and can be thought of as the variability that remains to be explained by the model.

F-statistic

The `F-statistic` is a test statistic for the following hypothesis test

$$\text{H}_0 : \beta = \vec{0}$$
$$\text{vs.}$$
$$\text{H}_1 : \beta_j \neq 0, \text{for at least one } j.$$

It tests if at least one of the model covariates have explanatory power. The test statistic is constructed as follows

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

where

$$\text{TSS} := \sum_{i=1}^{n} (y_i - \bar{y})^2$$

The test statistic is Fisher distributed with p and $n-p-1$ degrees of freedom, in our case 4 and 649 respectively. This test results in a p-value of less than $2.2 \cdot 10^{-16}$, and we can relatively confidently conclude that the regression is significant.

Q3:

The proportion of variability explained by `modelA` can be found by calculating the multiple R-squared statistic, given by

$$R^2 := \frac{\text{TSS} - \text{RSS}}{\text{TSS}} = 1 - \frac{\text{RSS}}{\text{TSS}} \in [0, 1]$$

TSS − RSS can be interpreted as the amount of variability explained by the model, while TSS is the total variability in the training set. In our case, this value is $\approx 80.95\%$.

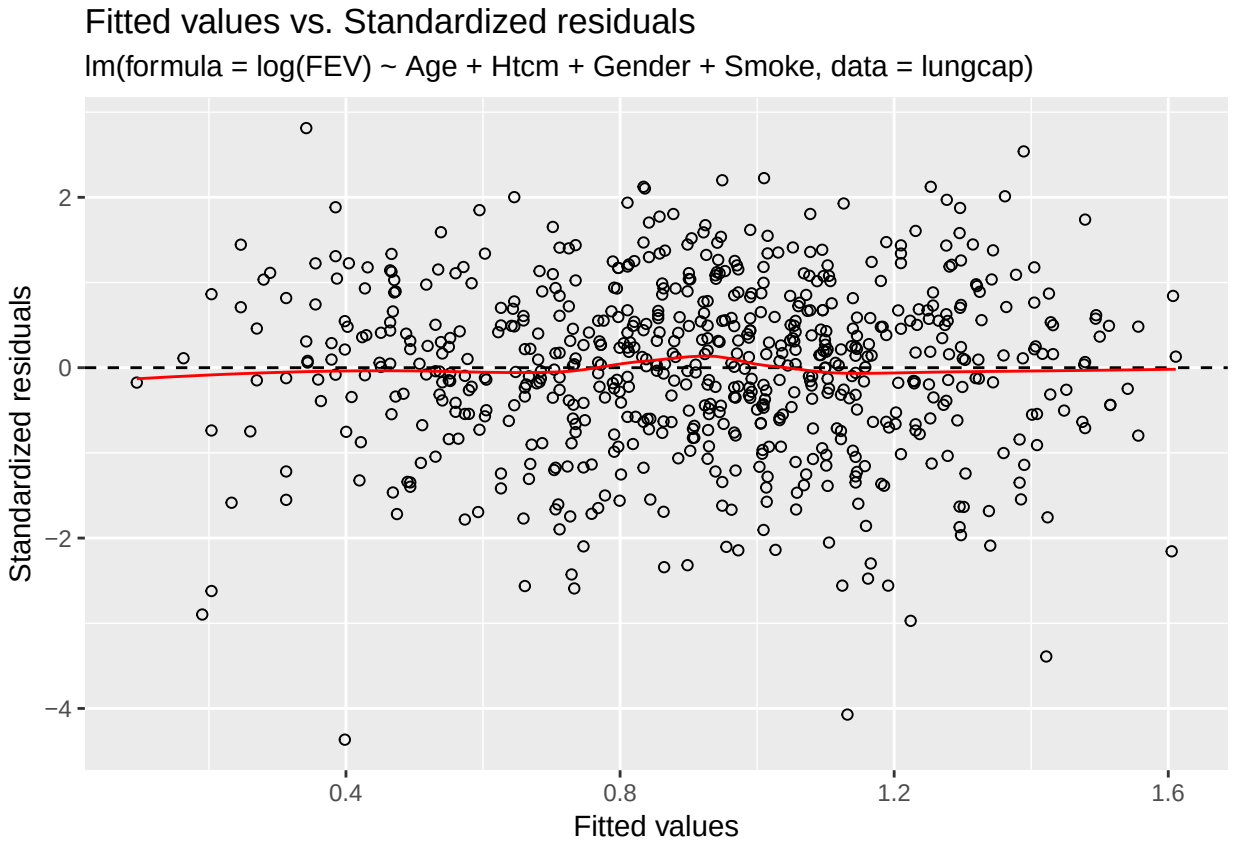
Q4:

Here we use *standardized residual plots* in order to check our model assumptions. One of our assumptions is

$$\epsilon \sim \mathrm{N}_n(0, \sigma^2 I).$$

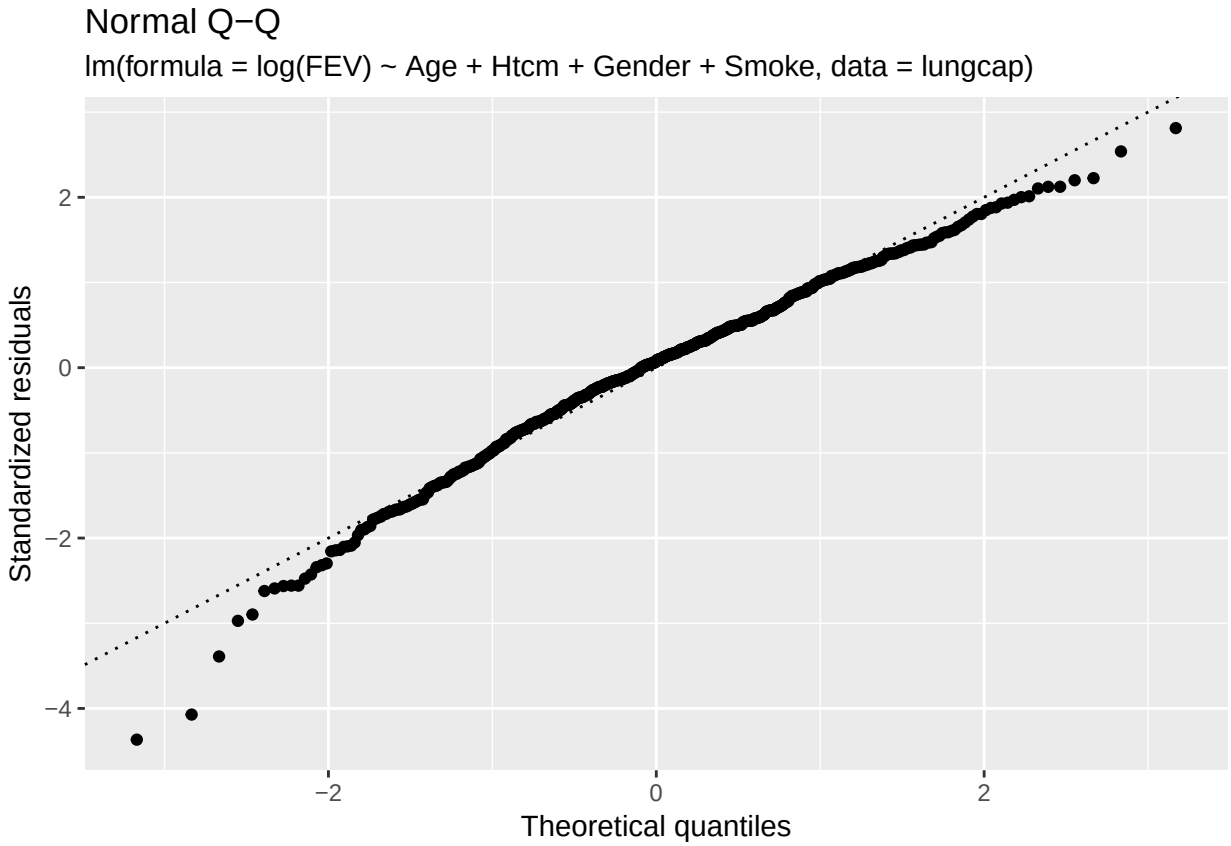
With other words, the error terms are identically, and independently normally distributed with mean zero. We can use the model standardized residuals as estimators for the error terms, and check these assumptions. First,

```
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)
  )
```



Here we expect the standardized residuals to be symmetrically distributed across the x-axis (zero expected), and no discernible magnitude trend along the x-axis. The latter requirement follows from the independence between the observational pairs (Y_i, x_i) and the error terms ϵ_i. Both requirements seem to be satisfied, so no issues so far. Next,

```
# 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)
  )
```



Here we plot a normal Q-Q plot for the model residuals. Ideally, we would like to see all points lying perfectly along a line going through the origin. In this case, we have some deviation from this straight line at the tails. This indicates a somewhat heavy left tail and light right tail, compared to the normal distribution.

Finally,

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

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

Here, the "Anderson-Darling normality test" is performed on the *studentized residuals*. Due to a large value for A, we must reject the null-hypothesis of residual normality. Since these results are not ideal, we should be somewhat careful while making inferences going forwards.

Q5:

We now fit a new model, `modelB`, identical to `modelA` except for not using a *logarithmic* transformation for `FEV`.

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

```
##
## Call:
## lm(formula = FEV ~ Age + Htcm + Gender + Smoke, data = lungcap)
```

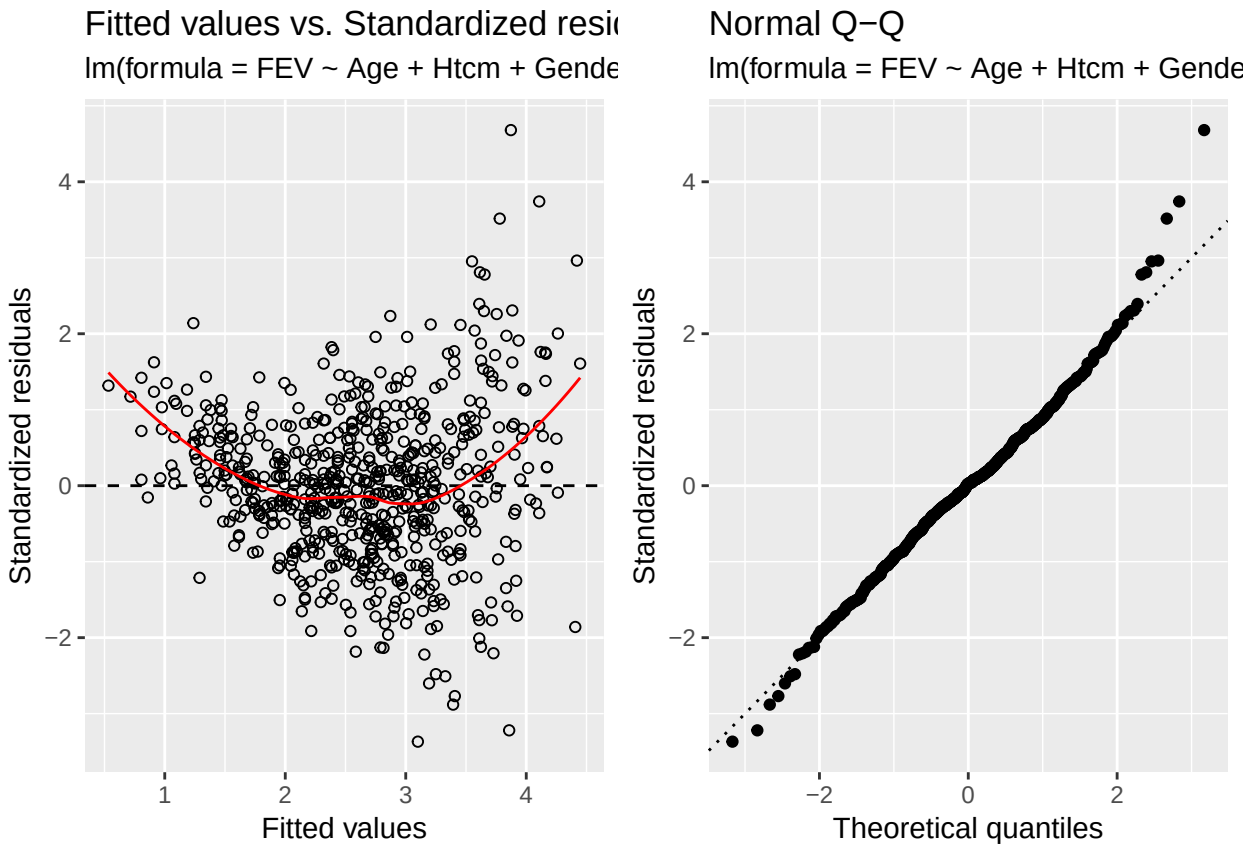
```
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.37656 -0.25033  0.00894  0.25588  1.92047
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -4.456974   0.222839 -20.001  < 2e-16 ***
## Age          0.065509   0.009489   6.904 1.21e-11 ***
## 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
```

Since we have the exact same covariates, we can assess the two models by comparing their R^2 statistics. Since $R^2_B < R^2_A$, `modelA` is to be preferred if explanatory power is important.

Arguably, we should still choose `modelB` if it satisfies the error term normality condition, since this is an important assumption of the linear regression model.

First, let's look at the residual plots, reusing the code from before

```
library(gridExtra)
plot1 <- 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$call)
  )
plot2 <- ggplot(modelB, aes(sample = .stdresid)) +
  stat_qq(pch = 19) +
  geom_abline(
    intercept = 0,
    slope = 1,
    linetype = "dotted"
  ) +
  labs(
    x = "Theoretical quantiles",
    y = "Standardized residuals",
    title = "Normal Q-Q",
    subtitle = deparse(modelB$call)
  )
grid.arrange(plot1, plot2, ncol = 2)
```



These plots show a clear trend between fitted values and the standardized residuals, much more so than `modelA`, while the Q-Q plot indicates a too light-tailed distribution. Lastly, let's repeat the Anderson-Darling normality test for `modelB`

```
ad.test(rstudent(modelB))
```

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

We can't conclude that error normality is satisfied here, so we still choose `modelA` over `modelB`.

Q6:

We now want perform the following hypothesis test

$$\mathrm{H}_0 : \beta_{\text{age}} = 0$$
$$\text{vs.}$$
$$\mathrm{H}_1 : \beta_{\text{age}} \neq 0.$$

In order to test this, define the following t-distributed test statistic

$$T_{\text{age}} = \frac{\hat{\beta}_{\text{age}} - \beta_{\text{age}}}{\hat{\mathrm{SE}}(\hat{\beta}_{\text{age}})} \sim t_{n-p-1}.$$

The p-value for the test is defined by

$$p\text{-value} := P(T_{\text{age}} > t_{\text{age}})$$

We now calculate this value

```
n <- nobs(modelA)
p <- length(modelA$coefficients)
betaAge <- beta["Age"]
betaSE <- modelAEstimates$`Std. Error`[2]
tTestStatistic <- betaAge / betaSE
pValue <- 2 * pt(q = tTestStatistic, df = n - p - 1, lower.tail = FALSE)
pValue
```

```
##          Age
## 7.105907e-12
```

The p-value is $\approx 7.1 \cdot 10^{-12}$, which coincides with the model summary output from earlier.

With a significance level of 0.01, the critical value (which gives the cut-off) can be compared with our t-statistic.

```
alpha <- 0.01
criticalValue <- qt(p = alpha / 2, df = n - p - 1, lower.tail = FALSE)
kable(tibble("t-statistic" = tTestStatistic, "critical value" = criticalValue))
```

t-statistic	critical value
6.984488	2.583438

We can reject the null-hypothesis and conclude that $\beta_{\text{age}} \neq 0$.

Q7:

A 99% confidence interval for β_{age} can be constructed with the same test statistic as above in the following way

$$\begin{aligned} P(-t_{\alpha/2,n-2} < T_{\text{age}} < t_{\alpha/2,n-2}) &= 1-\alpha \\ \implies P(-t_{\alpha/2,n-2} < \frac{\hat{\beta}_{\text{age}} - \beta_{\text{age}}}{\hat{\text{SE}}(\hat{\beta}_{\text{age}})} < t_{\alpha/2,n-2}) &= 1-\alpha \\ \implies P(\hat{\beta}_{\text{age}} - t_{\alpha/2,n-2} \cdot \hat{\text{SE}}(\hat{\beta}_{\text{age}}) < \beta_{\text{age}} < \hat{\beta}_{\text{age}} + t_{\alpha/2,n-2} \cdot \hat{\text{SE}}(\hat{\beta}_{\text{age}})) &= 1-\alpha \end{aligned}$$

So we must find the interval $\hat{\beta}_{\text{age}} \pm t_{\alpha/2,n-2} \cdot \hat{\text{SE}}(\hat{\beta}_{\text{age}})$.

```
alpha <- 0.01
radius <- qt(p = alpha / 2, df = n - p - 2, lower.tail = FALSE) * betaSE
confidenceInterval <- tibble(lower = betaAge - radius, upper = betaAge + radius)
confidenceInterval %>% kable()
```

lower	upper
0.0147367	0.0320378

A 99% confidence interval can be interpreted as an interval that has a 99% chance of covering the true parameter value, β_{age}. Notice that the 99% confidence interval for β_{age} is entirely positive. We can therefore relatively confidently conclude that `age` has a positive effect on `FEV`, as the multiplicative effect becomes greater than one, which is really useful. This positive 99% interval (i.e. not containing 0) is also closely related to the fact that we rejected the null-hypothesis at a 1% p-value cut-off in the previous task, since we can say with $\gtrapprox$ 99% confidence that β_{age} is nonzero.

Q8:

We now want to make a model prediction for a 16 year old non-smoking male of height 170 centimeters, x_0.

First we want to predict a value for `log(FEV)` by calculating

$$\widehat{\log(\text{FEV})} = x_0^T \hat{\beta}$$

This can be done with the R-function `predict()` like this

```
new = data.frame(Age=16, Htcm=170, Gender="M", Smoke=0)
logPrediction <- as_tibble(
  predict(
    modelA,
    newdata = new,
    interval = "predict",
    type = "response",
    level = 0.95
  )
) %>%
  rename(
    lower = lwr,
    upper = upr
  )
print(logPrediction$fit)
```

```
## [1] 1.323802
```

Our best guess, based on `modelA`, for `log(FEV)` is thus ≈ 1.32.

We can also construct a confidence interval for this predicted logarithmic value by using the following formula (given without proof)

$$[\mathbf{x}_0^T \hat{\beta} - t_{\alpha/2,n-p-1}\hat{\sigma}\sqrt{1+\mathbf{x}_0^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_0},\ \mathbf{x}_0^T \hat{\beta} + t_{\alpha/2,n-p-1}\hat{\sigma}\sqrt{1+\mathbf{x}_0^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_0}],$$

.

and exponentially transform the interval in order to get a 95% prediction interval for `FEV`. We can use `predict()` for this purpose as well

```
logInterval <- logPrediction[c("lower", "upper")]
FEVInterval <- exp(logInterval)
FEVInterval %>% kable()
```

lower	upper
2.818373	5.010039

Our 95% prediction interval is $[2.82, 5.01]$, which is a quite wide interval. If we compare this interval to our `FEV` observations

```
summary(lungcap$FEV)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.791   1.981   2.547   2.637   3.119   5.793
```

the most useful thing we can conclude is that the individual likely has an above average `FEV`.

Problem 2: Classification

```
library(class)# for function knn
library(caret)# 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:

Define $\mathcal{N}_x$ to be the set of the K closest points to the point $x = (x_1, x_2)$. The KNN istimator $\hat{y}(x) \in \{0, 1\}$ is determined by taking a "majority vote" of x's N closest neighbors. The KNN estimate of the posterior class probability becomes

$$\hat{P}(Y = y|X = x) = \frac{1}{K} \sum_{i \in \mathcal{N}_x} I(y_i = y) = \begin{cases} \frac{1}{K} \sum_{i \in \mathcal{N}_x} y_i, & \text{if } y = 1 \\ 1 - \frac{1}{K} \sum_{i \in \mathcal{N}_x} y_i, & \text{if } y = 0 \end{cases}$$

And the Bayes classifier can be used to construct the estimator $\hat{y}(x)$

$$\hat{y}(x) = \begin{cases} 1, & \text{if } \hat{P}(Y = 1|X = x) \geq 0.5 \\ 0, & \text{otherwise} \end{cases}$$

Q10:

The training error rate is given by

$$\frac{1}{n_{train}} \sum_{i \in J_{train}} I(y_i \neq \hat{y}(x_i)),$$

where J_{train} is the set of n_{train} indices defining the training set. Similarly the test error rate is given by

$$\frac{1}{n_{\text{test}}} \sum_{i \in J_{\text{test}}} I(y_i \neq \hat{y}(x_i))$$

We now calculate the error rates of the training and test sets for $K = 1, 2, ..., 30$ and plot these results.

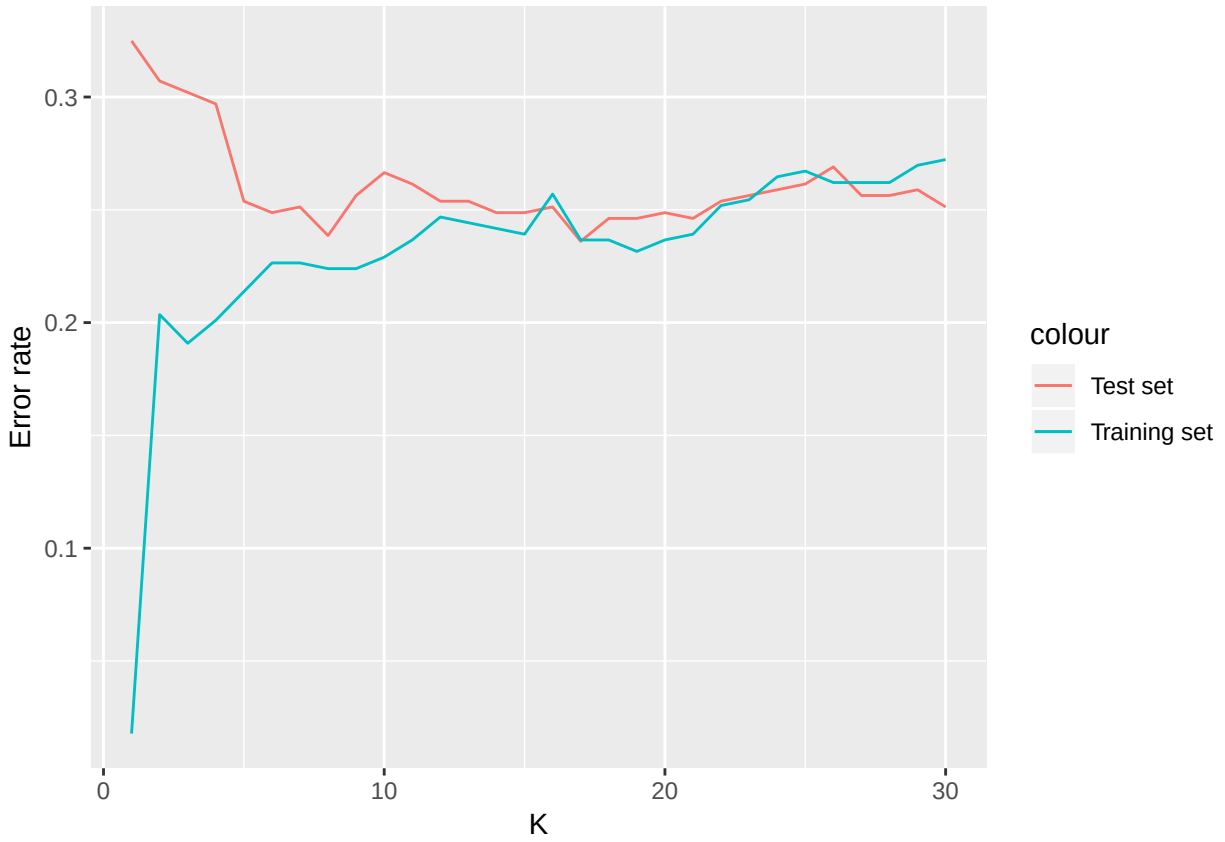
```
dataset <- as_tibble(Mdf)
train <- dataset %>% filter(istest == "0") %>% select(x1, x2)
test <- dataset %>% filter(istest == "1") %>% select(x1, x2)

trainAnswers <- dataset %>% filter(istest == "0") %>% pull(y)
testAnswers <- dataset %>% filter(istest == "1") %>% pull(y)
```

```r
train.e <- vector()
test.e <- vector()

knnErrorRates <- function(train, test, trainAnswers, testAnswers) {
  errorRates <- vector()
  for (k in 1:30) {
    testPredictions <- class::knn(
      train = train,
      test = test,
      cl = trainAnswers,
      k = k
    )
    errorRates <- append(
      errorRates,
      mean(testPredictions != testAnswers)
    )
  }
  return(errorRates)
}
test.e <- knnErrorRates(
  train = train,
  test = test,
  trainAnswers = trainAnswers,
  testAnswers = testAnswers
)
train.e <- knnErrorRates(
  train = train,
  test = train,
  trainAnswers = trainAnswers,
  testAnswers = trainAnswers
)
result <- tibble(
  K = 1:30,
  testError = test.e,
  trainError = train.e
)

result %>% ggplot(aes(x = K)) +
  geom_line(aes(y = testError, col = "Test set")) +
  geom_line(aes(y = trainError, col = "Training set")) +
  ylab("Error rate")
```



As expected, the training set has the lowest error rate for $K = 1$, since the model closely fits the training data. The reason for the error not being exactly zero at $K = 1$ is due to duplicate x entries.

Q11:

We now calculate the cross-validation (CV) error using five folds.

```
set.seed(0)
# Choose K from 1 to 30.
ks <- 1:30
# Divide the training data into 5 folds.
idx <- createFolds(M[tr,1], k=5)
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)
      }
    )
  }
)
```

Now we can calculate the average CV error, the standard error of the average CV error over the five folds, and the K-value which corresponds to the smallest CV error.

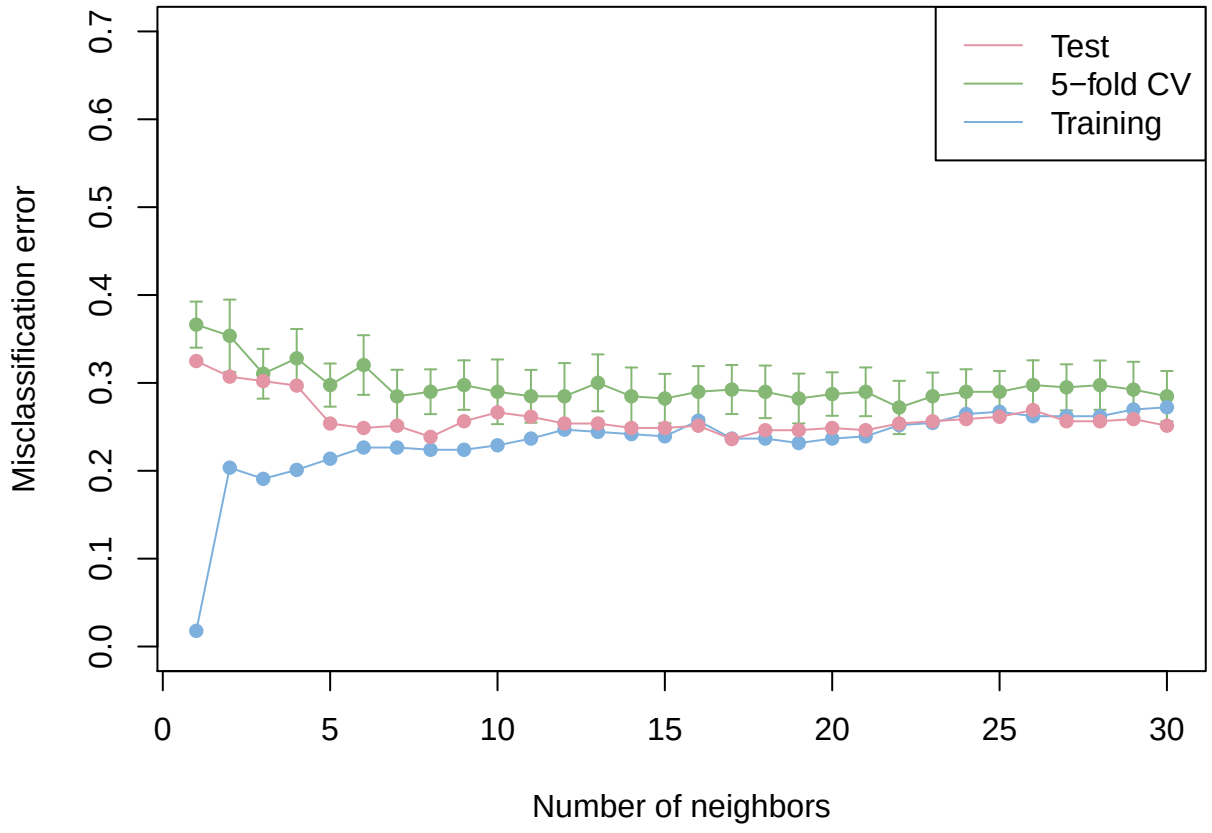
```
cv.e <- colMeans(cv)
cv.se <- apply(cv, 2, sd) / sqrt(5)
k.min <- which.min(cv.e)
print(k.min)
```

```
## [1] 22
```

Q12:

We now plot the misclassification errors.

```
library(colorspace)
co <- rainbow_hcl(3)
par(mar = c(4,4,1,1) + 0.1, 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)
```

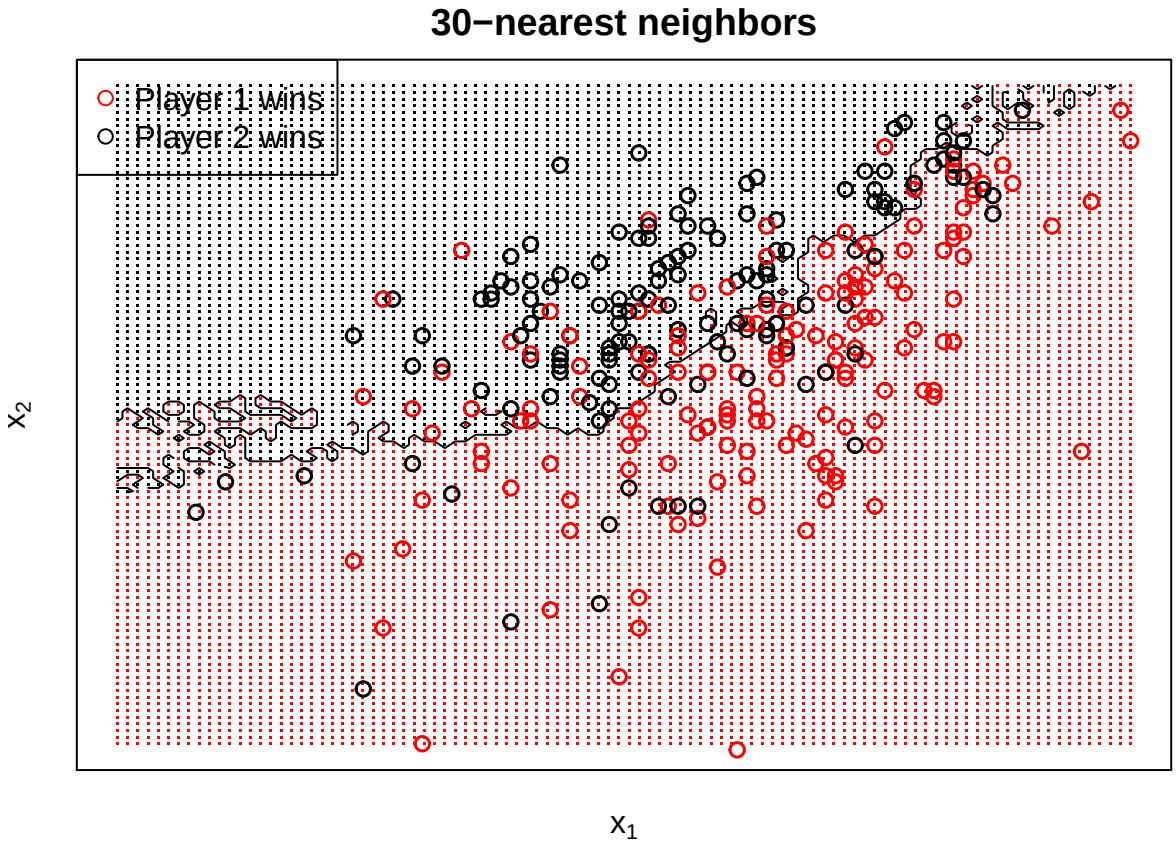


The bias of $\hat{y}(x)$ will increase with increasing K, but the variance will *decrease* with increasing K. The variance is high for $K = 1$ because only the nearest neighbours is used and noise will therefore affect the model quite a lot. The bias is low for $K = 1$ because the model predictions will be very close to the training data.

Q13:

A different strategy will now be used to choose K.

```
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 line of interest is

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

Instead of choosing the K that results in the smallest CV error rate, $\hat{\mathrm{K}} := \operatorname{argmin} \mathrm{CV}(\mathrm{K})$, we instead choose the greatest K which is within one standard deviation of $\hat{\mathrm{K}}$. More exactly,

$$\mathrm{K} = \max_{\mathrm{K}^*} \{K^* \mid \mathrm{CV}(\mathrm{K}^*) \leq \mathrm{CV}(\hat{\mathrm{K}}) + \mathrm{SE}(\hat{\mathrm{K}})\}$$

This way, we are able to simplify our model within acceptable bounds from the minimal error.

Q14:

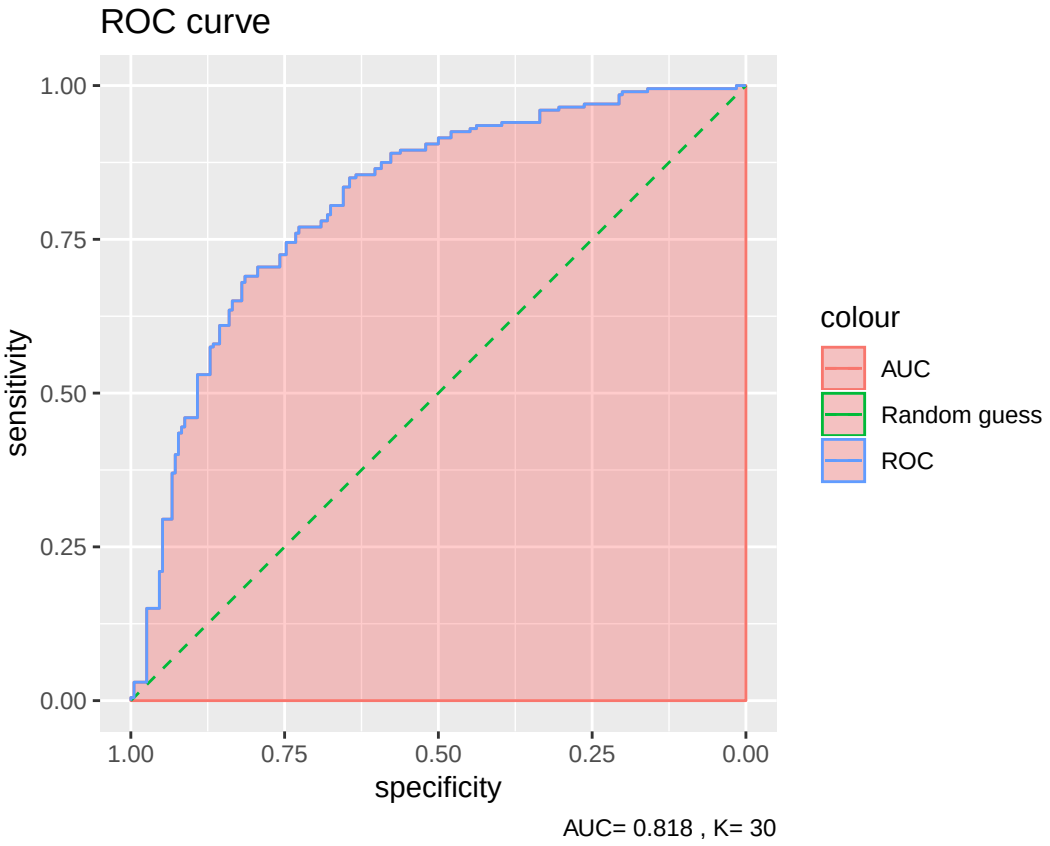
We now plot the receiver operating characteristics (ROC) curve.

```
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)
player1ROC <- roc(response = M[-tr, 1], predictor = KNNprob, legacy.axes = TRUE)
ROC <- tibble(
  sensitivity = rev(player1ROC$sensitivities),
  specificity = rev(player1ROC$specificities)
)

# For stepribbon functionality
library(ggalt)
ROC %>% ggplot() +
  geom_ribbon(
    aes(
      x = specificity,
      ymin = 0,
      ymax = sensitivity,
      col = "AUC"
    ),
    fill = "red",
    alpha = 0.2,
    stat = "stepribbon"
  ) +
  geom_line(
    data = tibble(x = c(1, 0), y = c(0, 1)),
    aes(x = x, y = y, col = "Random guess"),
    linetype = "dashed"
  ) +
  geom_step(
    aes(x = specificity, y = sensitivity, col = "ROC")
  ) +
  labs(
    title = "ROC curve",
    caption = paste(
      c("AUC=", signif(player1ROC$auc, 3), ", K=", K),
      collapse = " "
    )
  ) +
  scale_x_reverse() +
  coord_equal() +
  ylab("sensitivity")
```



This curve shows the relationship between the *specificity* and *sensitivity* of the model as the threshold probability goes from 0 to 1, defined as

$$\text{sensitivity} := \frac{\text{Number of true positives}}{\text{Number of positives}} = \frac{\text{TP}}{\text{P}}$$
$$\text{specificity} := \frac{\text{Number of true negatives}}{\text{Number of negatives}} = \frac{\text{TN}}{\text{N}}$$

Thus, these extreme thresholds (probability cut-off) must therefore imply the following

$$\text{threshold} \leftarrow 0 \implies \begin{cases} \text{sensitivity} = 1 \\ \text{specificity} = 0 \end{cases}$$
$$\text{threshold} \leftarrow 1 \implies \begin{cases} \text{sensitivity} = 0 \\ \text{specificity} = 1 \end{cases}$$

The AUC $\in [0, 1]$ is the area under the ROC curve. The best case scenario is when the ROC curve "hugs" the upper left corner, resulting in AUC $= 1$. In our case, we have AUC ≈ 0.818.

Random guessing can be thought of as drawing samples $u \sim \mathcal{U}(0, 1)$ and comparing it to the threshold probability in order to make a prediction. Since $P(u < p) = p$, the resulting specificity-sensitivity graph becomes a straight line (on average), and the AUC becomes 0.5.

A random guess classifier corresponds to a posterior distribution $\text{P}(Y = 1|X = x) \sim \mathcal{U}(0, 1)$. This in turn means that $\text{TP} \sim \mathcal{B}\rangle\backslash(P, 1 - t)$, t being the threshold, and $\text{TN} \sim \mathcal{B}\rangle\backslash(N, t)$. Thus

$$\text{E[sensitivity]} = \text{E}\left[\frac{TP}{P}\right] = 1 - t = 1 - \text{E}\left[\frac{TN}{N}\right] = 1 - \text{E[specificity]}$$

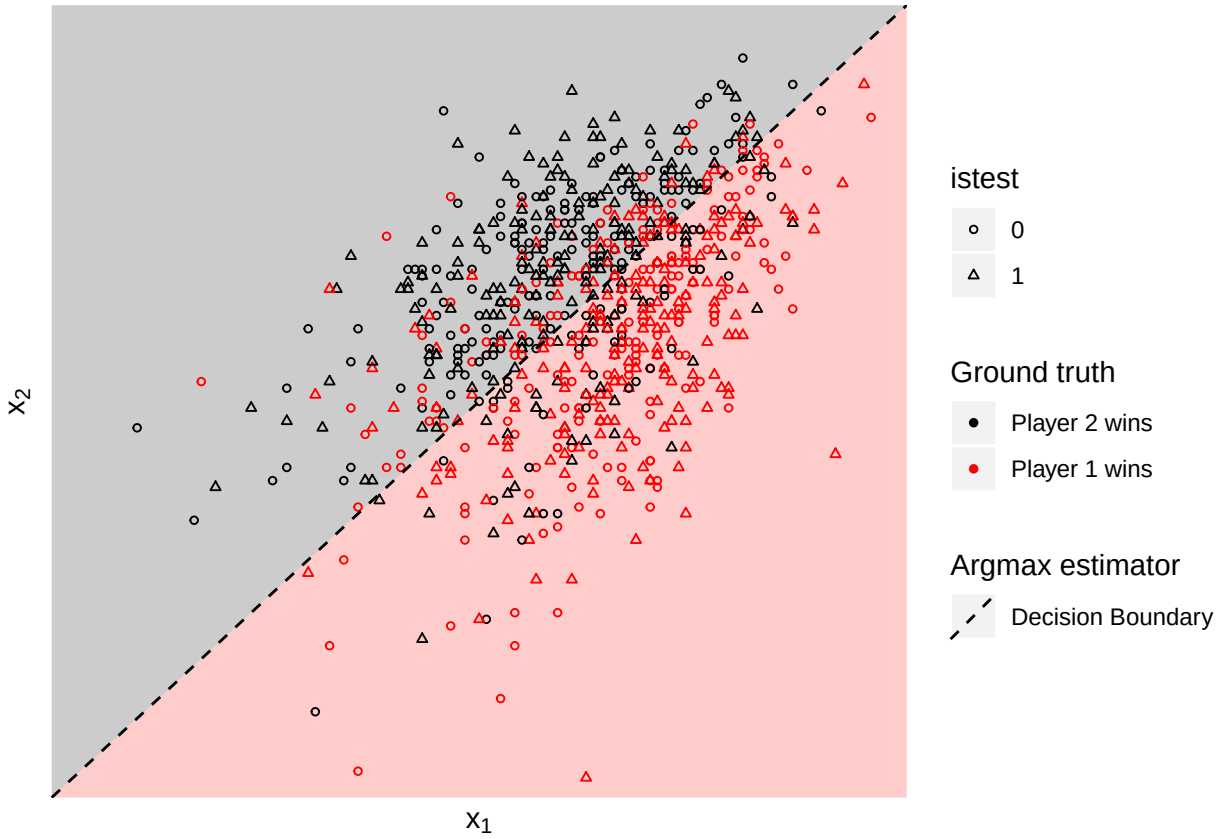
Which means that the ROC curve will be expected to lie on the green diagonal in the plot, which will result in $\text{E[AUC]} = 0.5$

Q15:

We now plot the decision boundary of $\tilde{y}(x) = \text{argmax}_k$, which is simply the line $x_1 = x_2$.

```
tb <- tibble(
  x1 = M$x1,
  x2 = M$x2,
  y = M$y,
  yHat = as.integer(M$x1 > M$x2),
  istest = as.factor(trte)
)
p1WinsPoly <- tibble(
  x1 = c(-100, 20, 20),
  x2 = c(-100, -100, 20)
)
p2WinsPoly <- tibble(
  x1 = c(-100, 20, -100),
  x2 = c(-100, 20, 20)
)
tb %>% ggplot() +
  aes(x = x1, y = x2) +
  geom_polygon(
    data = p1WinsPoly,
    aes(x = x1, y = x2),
    alpha = 0.2,
    fill = "red"
  ) +
  geom_polygon(
    data = p2WinsPoly,
    aes(x = x1, y = x2),
    alpha = 0.2,
    fill = "black"
  ) +
  geom_point(
    aes(col = y, x = x1, y = x2, shape = istest),
    size = 1
  ) +
  geom_abline(
    aes(
      slope = 1,
      intercept = 0,
      linetype = "Decision boundary"
    )
  ) +
  xlab(expression("x"[1])) +
  ylab(expression("x"[2])) +
  scale_color_manual(
    name = "Ground truth",
    labels = c("Player 2 wins", "Player 1 wins", ""),
```

```
    values = c("black", "red", "gray")
  ) +
  scale_linetype_manual(
    name = "Argmax estimator",
    labels = "Decision Boundary",
    values = "dashed"
  ) +
  scale_shape_manual(values = c(1, 2)) +
  scale_x_continuous(limits = c(-100, 20), expand = c(0, 0)) +
  scale_y_continuous(limits = c(-100, 20), expand = c(0, 0)) +
  theme(
    axis.ticks = element_blank(),
    axis.text = element_blank(),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.background = element_blank()
  )
```



Now we move onto the calculation of the confusion matrices and misclassification errors of $\hat{y}(x)$ and $\tilde{y}(x)$.

```
train <- tb %>% slice(-tr)
argmaxConfMatrix <- confusionMatrix(table(train$yHat, train$y))
knnConfMatrix <- confusionMatrix(table(KNNclass, train$y))
print(argmaxConfMatrix)
print(knnConfMatrix)
```

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

These two predictors perform really similarly, since the 95% confidence interval for the accuracy is quite overlapping. The KNN predictor has better specificity, while the argmax predictor has better sensitivity. Neither specificity nor sensitivity is of particual interest, since the choice of player 1 is arbitrary, and it is a

"symmetric" problem. Hence, we prefer the argmax predictor since it has better overall accuracy.

Problem 3: Bias-variance trade-off

Q16:

Consider the least squares estimator for a linear regression model

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

First, let's find the expected value for $\hat{\beta}$. Here we will use that a linear combination, C, of a normal vector, Y, satisfies $E[CY] = CE[Y]$, and that $E[\epsilon] = 0$:

$$\begin{aligned} E[\hat{\beta}] &= E\left[(X^TX)^{-1}X^TY\right] = (X^TX)^{-1}X^TE[Y] \\ &= (X^TX)^{-1}X^TE[X\beta + \epsilon] = (X^TX)^{-1}X^TX\beta = \beta \end{aligned}$$

This is an unbiased estimator.

Now onto the variance-covariance matrix for $\hat{\beta}$. Here, we will use the fact that $\mathrm{Cov}(CY) = C\mathrm{Cov}(Y)C^T$, and $\mathrm{Cov}(Y) = \sigma^2 I$.

$$\begin{aligned} \mathrm{Cov}(\hat{\beta}) &= \mathrm{Cov}\left((X^TX)^{-1}X^TY\right) = (X^TX)^{-1}X^T\mathrm{Cov}(Y)((X^TX)^{-1}X^T)^T \\ &= (X^TX)^{-1}X^T\sigma^2 I((X^TX)^{-1}X^T)^T = (X^TX)^{-1}\sigma^2 \end{aligned}$$

So the result is that $\hat{\beta} \sim N_{k+1}(\beta, \sigma^2(X^TX)^{-1})$, k being the number of covariates in the model.

Q17:

Now, we find the expected value of $\hat{f}(x_0)$.

$$E\left[\hat{f}(x_0)\right] = E\left[x_0^T\hat{\beta}\right] = x_0^TE[\hat{\beta}] = x_0^T\beta = E[Y_0]$$

So this is also an unbiased predictor.

Now, the variance of the predictor is

$$\mathrm{Var}(\hat{f}(x_0)) = \mathrm{Var}(x_0^T\hat{\beta}) = x_0^T\mathrm{Cov}(\hat{\beta})x_0 = \sigma^2 x_0^T(X^TX)^{-1}x_0.$$

So, we have $\hat{f}(x_0) \sim N_1(x_0^T\beta,\ \sigma^2 x_0^T(X^TX)^{-1}x_0)$.

Q18:

We will use that for a random variable, Z, we have $\mathrm{E}\left[Z^2\right] = \mathrm{Var}(Z) + \mathrm{E}[Z]^2$. Also, since Y_0 and $\hat{f}(x_0)$ are independent, then $\mathrm{E}[Y_0\hat{f}(x_0)] = \mathrm{E}[Y_0]\mathrm{E}[\hat{f}(x_0)]$.

$$
\begin{aligned}
&\mathrm{E}[(Y_0 - \hat{f}(\mathbf{x}_0))^2] \\
&= \mathrm{E}\left[Y_0^2 + \hat{f}(x_0)^2 - 2Y_0\hat{f}(x_0)\right] \\
&= \mathrm{E}\left[Y_0^2\right] + \mathrm{E}\left[\hat{f}(x_0)^2\right] - \mathrm{E}\left[2Y_0\hat{f}(x_0)\right] \\
&= \mathrm{Var}(Y_0) + \mathrm{E}\left[Y_0\right]^2 + \mathrm{Var}\left(\hat{f}(x_0)\right) + \mathrm{E}\left[\hat{f}(x_0)\right]^2 - 2 \cdot \mathrm{E}\left[Y_0\right]\mathrm{E}\left[\hat{f}(x_0)\right] \\
&= \mathrm{Var}(Y_0) + f(x_0)^2 + \mathrm{Var}\left(\hat{f}(x_0)\right) + \mathrm{E}\left[\hat{f}(x_0)\right]^2 - 2 \cdot f(x_0)\mathrm{E}\left[\hat{f}(x_0)\right] \\
&= \mathrm{Var}(Y_0) + \mathrm{Var}\left(\hat{f}(x_0)\right) + \left(\mathrm{E}\left[\hat{f}(x_0)\right] - f(x_0)\right)^2 \\
&= \sigma^2 I + \sigma^2 x_0^T (X^T X)^{-1} x_0 + (x_0^T\beta - x_0^T\beta)^2 \\
&= \sigma^2 I + \sigma^2 x_0^T (X^T X)^{-1} x_0 + 0^2 \\
&= \text{Irreducible error} + \text{Variance} + \text{Bias}^2
\end{aligned}
$$

The bias becomes zero as we have perfect knowledge about the true function f and a model that reflects this.

Q19:

We now look at the ridge estimator, $\widetilde{\beta}$,

$$\widetilde{\beta} := (X^T X + \lambda I)^{-1} X^T Y, \quad \lambda \in [0, \infty]$$

First, we calculate the expected value.

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

Since $\widetilde{\beta} \neq \hat{\beta}$, this is a biased estimator!

The variance becomes

$$
\begin{aligned}
&\mathrm{Var}(\widetilde{\beta}) = \mathrm{Var}\left((X^T X + \lambda I)^{-1} X^T Y\right) \\
&= (X^T X + \lambda I)^{-1} X^T \ \mathrm{Var}(Y) \ \left((X^T X + \lambda I)^{-1} X^T\right)^T \\
&= \sigma^2 (X^T X + \lambda I)^{-1}(X^T X)(X^T X + \lambda I)^{-1}.
\end{aligned}
$$

So we end up with $\widetilde{\beta} \sim N_{k+1}\left((X^T X + \lambda I)^{-1}(X^T X)\beta,\ \sigma^2 (X^T X + \lambda I)^{-1}(X^T X)(X^T X + \lambda I)^{-1}\right)$.

Q20:

Now we find the expected value of the estimator $\widetilde{f}(x_0) = x_0^T\widetilde{\beta}$.

$$\mathrm{E}\left[\widetilde{f}(x_0)\right] = \mathrm{E}\left[x_0^T\widetilde{\beta}\right] = x_0^T \mathrm{E}\left[\widetilde{\beta}\right] = x_0^T (X^T X + \lambda I)^{-1}(X^T X)\beta$$

And the variance.

$$\mathrm{Var}\left(\widetilde{f}(x_0)\right) = \mathrm{Var}\left(x_0^T\widetilde{\beta}\right) = x_0^T \mathrm{Cov}\left(\widetilde{\beta}\right) x_0 = \sigma^2 x_0^T (X^T X + \lambda I)^{-1}(X^T X)(X^T X + \lambda I)^{-1} x_0$$

And therefore we have $\widetilde{f}(x_0) \sim N_1\left(x_0^T (X^T X + \lambda I)^{-1}(X^T X)\beta,\ \sigma^2 x_0^T (X^T X + \lambda I)^{-1}(X^T X)(X^T X + \lambda I)^{-1} x_0\right)$.

Q21:

Again, we decompose the MSE, skipping the initial decomposition proof this time

$$\begin{aligned}
&\mathrm{E}[(Y_0 - \widetilde{f}(\mathbf{x}_0))^2] \\
&= \left(\mathrm{E}\left[\widetilde{f}(x_0)\right] - f(x_0)\right)^2 + \mathrm{Var}\left(\widetilde{f}(\mathbf{x}_0)\right) + \mathrm{Var}(\varepsilon) \\
&= \left(x_0^T(X^TX + \lambda I)^{-1}(X^TX)\beta - x_0^T\beta\right)^2 + \sigma^2 x_0^T(X^TX + \lambda I)^{-1}(X^TX)(X^TX + \lambda I)^{-1}x_0 + \sigma^2 I \\
&= \mathrm{Bias}^2 + \mathrm{Variance} + \mathrm{Irreducible\ error}
\end{aligned}$$

We have therefore introduced a nonzero bias compared to the classical linear model.

Q22:

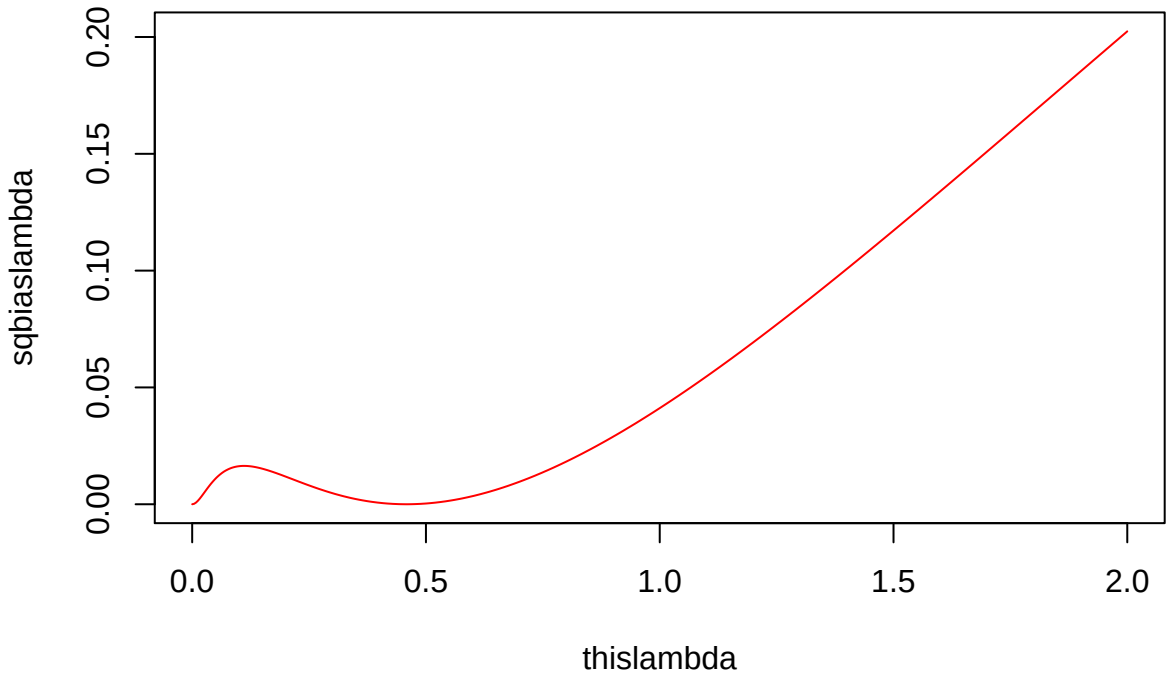
We will now fetch pre-calculated values for X, x_0, β and σ.

```
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
```

And plot the squared bias of $\widetilde{f}(x_0)$ as a function of λ.

```
sqbias <- function(lambda, X, x0, beta) {
  p <- dim(X)[2]
  XtX <- t(X) %*% X
  lambdaI <- diag(x = lambda, nrow = p)
  value <- (t(x0) %*% solve(XtX + lambdaI) %*% XtX %*% 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")
```

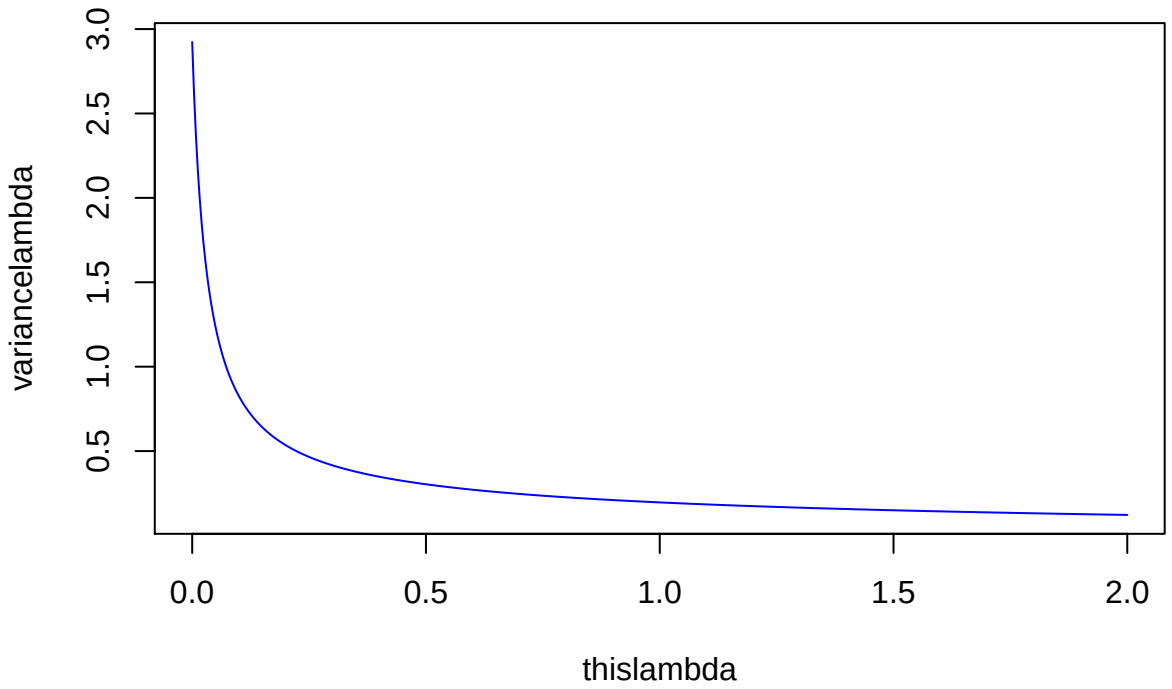


Since λ decreases the flexibility of the resulting model, we would expect the squared bias to increase as λ increases. Asymptotically, the graph reflects this prediction, but there is a "dip" around the region $[0.1, 0.5]$, which is not exactly what we expected.

Q23:

We now plot the variance of $\widetilde{f}(x_0)$ as a function of λ.

```
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")
```

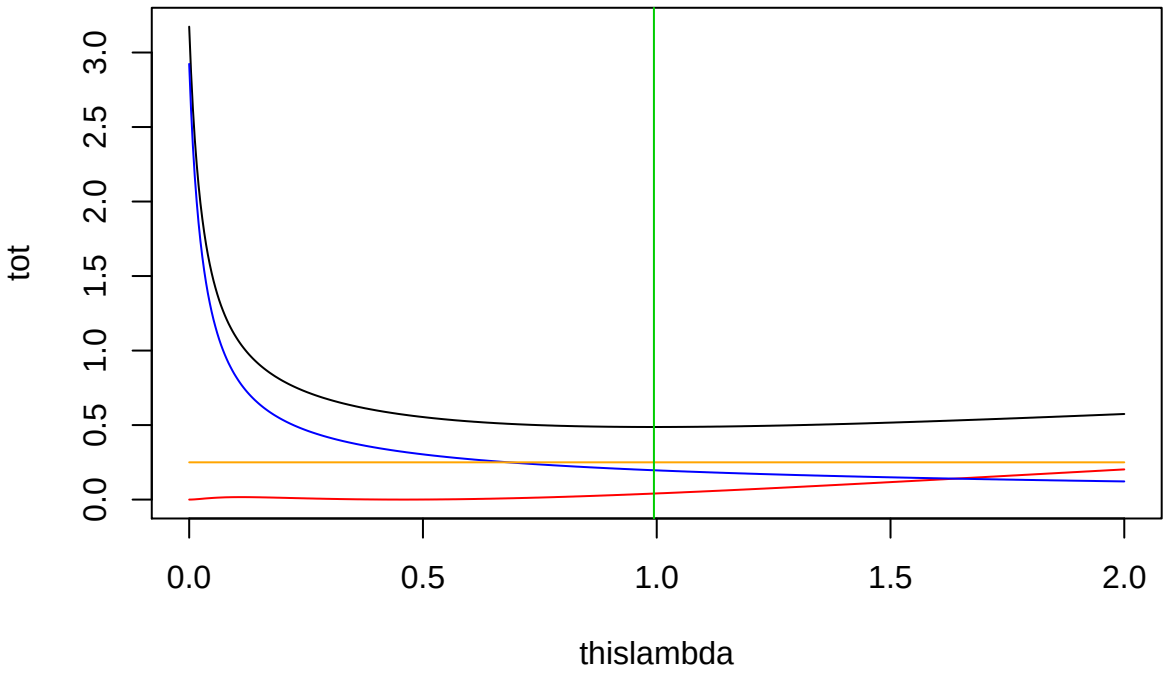


The plot looks like what we would expect, from the fact that as λ increases, then the diagonal λ matrix becomes more dominant relative to the test data. This is in accordance with the bias-variance trade-off, since the flexibility here decreases with increasing λ.

Q24:

We now plot all the MSE decompositions and the total MSE in the same figure.

```
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 λ is the one which minimizes the total test MSE, which in this case is $\lambda \approx 0.994$.