Compulsory exercise 1: Group 12

TMA4268 Statistical Learning V2019

Emma Skarnes, Håkon Noren and Alexander Johan Arntzen

20 February, 2020

Problem 1

For this problem you will need to include some LaTex code. Please install latex on your computer and then consult Compulsor1.Rmd for hints how to write formulas in LaTex.

a)

The expected test mean squared error (MSE) at x_0 is

$$E[y_0 - \hat{f}(x_0)]^2.$$

Where y_0 is the new observation.

b)

Firstly, from computation it is clear that

$$\begin{aligned}
\mathrm{E}[f(x_0) - \hat{f}(x_0)]^2 &= \mathrm{E}[f(x_0) - \mathrm{E}[\hat{f}(x_0)] + \mathrm{E}[\hat{f}(x_0)] - \hat{f}(x_0)]^2 \\
&= \mathrm{E}[f(x_0) - \mathrm{E}[\hat{f}(x_0)]]^2 + 2\mathrm{E}[f(x_0) - \mathrm{E}[\hat{f}(x_0)]\ ][\mathrm{E}[\hat{f}(x_0)] - \hat{f}(x_0)] + \mathrm{E}[\mathrm{E}[\hat{f}(x_0)] - \hat{f}(x_0)]^2 \\
&= (f(x_0) - \mathrm{E}[\hat{f}(x_0)])^2 \ + \mathrm{Var}[\hat{f}(x_0)].
\end{aligned}$$

Secondly note that y_0 and the training data are independent. In addition $\mathrm{E}[\epsilon] = 0$. Then, expanding the test~MSE, the decomposition becomes

$$\begin{aligned}
\mathrm{E}[y_0 - \hat{f}(x_0)]^2 \ &= \mathrm{E}[f(x_0) + \epsilon - \hat{f}(x_0)]^2 \\
&= \mathrm{E}[\epsilon - \mathrm{E}[\epsilon]]^2 + 2\mathrm{E}[\epsilon]\mathrm{E}[f(x_0) - \hat{f}(x_0)] + \mathrm{E}[f(x_0) - \hat{f}(x_0)]^2 \\
&= \mathrm{Var}[\epsilon] + [f(x_0) - \mathrm{E}[\hat{f}(x_0)]]^2 + \mathrm{Var}[\hat{f}(x_0)].
\end{aligned}$$

c)

Since variance and squared expression cannot be negative each of the three terms contributes to the expected test MSE at x_0.

The **Var[ϵ]** is the **irreducible error**. It is independent of the model and cannot be reduced.

Var[$\hat{f}(x_0)$] is the **variance** of the prediction at x_0. It is a measure of how much the model will change based on different training data. In general the variance of the model increases when the the flexibility of the model increases.

$[f(x_0) - \mathrm{E}[\hat{f}(x_0)]]^2$ is the **squared bias** at x_0, also denoted $\mathrm{Bias}(\hat{f}(x_0))^2$. It is a measure of how much the model differs from the target function f at x_0. In general a flexible model will have low bias.

A good model will strike a balance between bias and variance to achieve a low total expected test MSE.

d)

TRUE, FALSE, FALSE, TRUE

e)

TRUE, FALSE, TRUE, FALSE.

f)

(ii)

g)

C

Problem 2

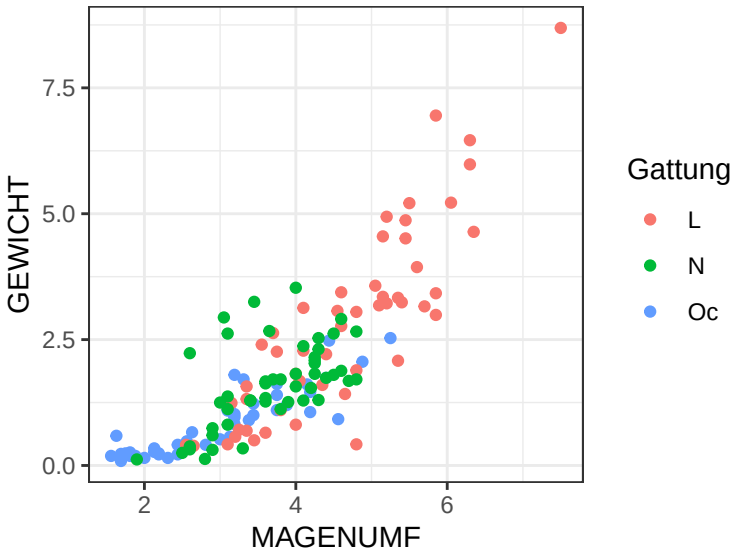
```
##   Gattung Nummer GEWICHT FANGDATUM MAGENUMF
## 1      Oc     32    0.19  23.09.97     1.56
## 2      Oc     34    0.59  23.09.97     1.63
## 3      Oc     48    0.09  23.09.97     1.69
## 4      Oc     55    0.23  23.09.97     1.69
## 5      Oc     41    0.24  23.09.97     1.75
## 6      Oc     24    0.19  23.09.97     1.81
```

a)

The dataset consists of 143 rows and 5 columns, which can be seen from the `str()`-function. The qualitative variables are Gattung and Nummer, while the quantitative variables are GEWICHT, FANGDATUM and MAGENUMF.

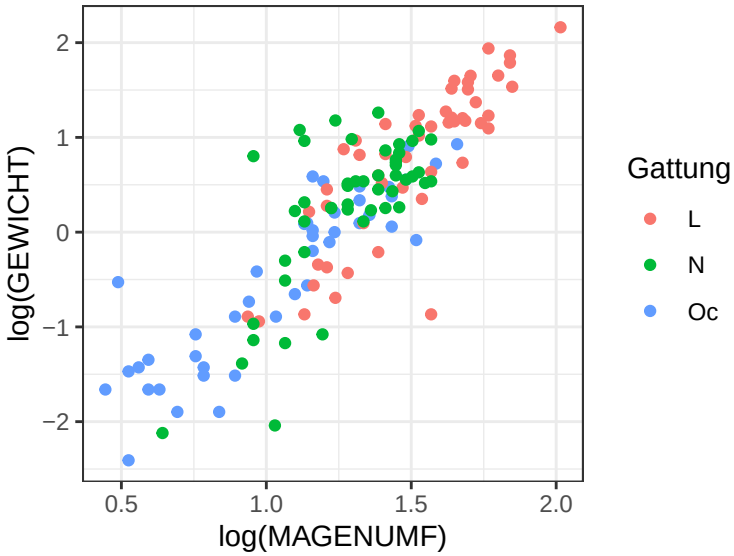
b)

```
ggplot(d.worm, aes(x = MAGENUMF, y = GEWICHT, colour = Gattung)) + geom_point() +
    theme_bw()
```



The relationship in this scatterplot does not look linear, so we linearize it by transforming the quantitative variables. Here, we take the logarithm of both:

```
ggplot(d.worm, aes(x = log(MAGENUMF), y = log(GEWICHT), colour = Gattung)) + geom_point() +
    theme_bw()
```



The relationship now looks quite linear, which can be seen more clearly by adding `+ geom\_smooth(method = "lm")` to the `ggplot()`-function. Especially the species denoted by L and Oc are close to the fitted line, while there is a bigger spread for the species denoted by N.

c)

```
fit.lm = lm(log(GEWICHT) ~ log(MAGENUMF) + Gattung, data = d.worm)
summary(fit.lm)
```

```
##
```

```
## Call:
## lm(formula = log(GEWICHT) ~ log(MAGENUMF) + Gattung, data = d.worm)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.79642 -0.25996  0.02864  0.25159  1.40557
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -3.13865    0.24106 -13.020   <2e-16 ***
## log(MAGENUMF)  2.59310    0.15378  16.862   <2e-16 ***
## GattungN       0.07327    0.10252   0.715    0.476
## GattungOc     -0.06149    0.12219  -0.503    0.616
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4832 on 139 degrees of freedom
## Multiple R-squared:  0.7598, Adjusted R-squared:  0.7547
## F-statistic: 146.6 on 3 and 139 DF,  p-value: < 2.2e-16
```

```
anova(fit.lm)
```

```
## Analysis of Variance Table
##
## Response: log(GEWICHT)
##                 Df  Sum Sq Mean Sq  F value Pr(>F)
## log(MAGENUMF)    1 102.264 102.264 438.0809 <2e-16 ***
## Gattung          2   0.394   0.197   0.8441 0.4321
## Residuals      139  32.448   0.233
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The separate equations of the regression models for the three different species are on the form

$$\log(\text{GEWICHT}) = \hat{\beta}_0 + \hat{\beta}_1 \cdot \log(\text{MAGENUMF}) + \hat{\beta}_2 x_{i1} + \hat{\beta}_3 x_{i2} + \epsilon_i,$$

where x_{i1} and x_{i2} are the dummy variables, given by

$$x_{i1} = \begin{cases} 1 & \text{if the ith worm is a Nicodrilus} \\ 0 & \text{if the ith worm is not a Nicodrilus} \end{cases}$$

$$x_{i2} = \begin{cases} 1 & \text{if the ith worm is an Octolasion} \\ 0 & \text{if the ith worm is not an Octolasion} \end{cases}$$

This gives the following equations:

$$\log(\text{GEWICHT}) = \begin{cases} -3.13865 + 2.59310 \cdot \log(\text{MAGENUMF}) + 0.24106 & \text{if Gattung="L"} \\ -3.13865 + 2.59310 \cdot \log(\text{MAGENUMF}) + 0.07327 + 0.24106 & \text{if Gattung="N"} \\ -3.13865 + 2.59310 \cdot \log(\text{MAGENUMF}) - 0.06149 + 0.24106 & \text{if Gattung="Oc"} \end{cases}$$

By looking at the outputs of the `summary()` and `anova()` functions, we observe that the p-values for the species are relatively high in the summary, which indicates that there is no statistical evidence of a difference

between the average weight of the three species - even though it might look like that from our plots. The anova output has an F-statistic of 0.8441 and associated p-value of 0.4321, which indicates that Gattung is not a relevant predictor.

d)

```
# Test interaction term
fit.lmInter = lm(log(GEWICHT) ~ log(MAGENUMF) * Gattung, data = d.worm)
summary(fit.lmInter)
```

```
##
## Call:
## lm(formula = log(GEWICHT) ~ log(MAGENUMF) * Gattung, data = d.worm)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.8114 -0.2830  0.0185  0.2457  1.4483
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)              -3.49061    0.42376  -8.237 1.25e-13 ***
## log(MAGENUMF)             2.82702    0.27804  10.168  < 2e-16 ***
## GattungN                  0.19491    0.61075   0.319    0.750
## GattungOc                 0.52368    0.48810   1.073    0.285
## log(MAGENUMF):GattungN   -0.05431    0.43824  -0.124    0.902
## log(MAGENUMF):GattungOc -0.45640    0.35462  -1.287    0.200
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4831 on 137 degrees of freedom
## Multiple R-squared:  0.7633, Adjusted R-squared:  0.7547
## F-statistic: 88.36 on 5 and 137 DF,  p-value: < 2.2e-16
```

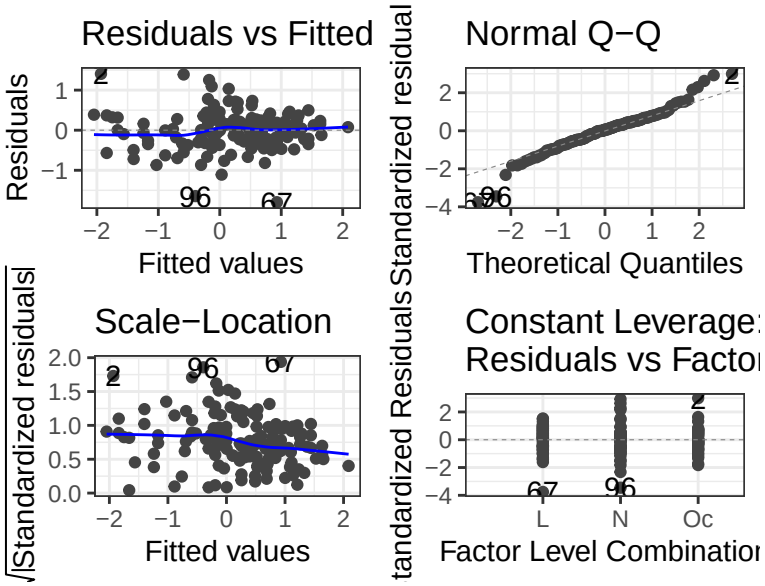
```
anova(fit.lmInter, fit.lm)
```

```
## Analysis of Variance Table
##
## Model 1: log(GEWICHT) ~ log(MAGENUMF) * Gattung
## Model 2: log(GEWICHT) ~ log(MAGENUMF) + Gattung
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1    137 31.978
## 2    139 32.448 -2  -0.46928 1.0052 0.3686
```

Here, the p-value are also relatively far away from zero, and the F-statistic from the anova table is close to 1, both of which supports our results from **2c)**. In other words, an interaction term between the logarithm of the circumference and the species does not improve the model.

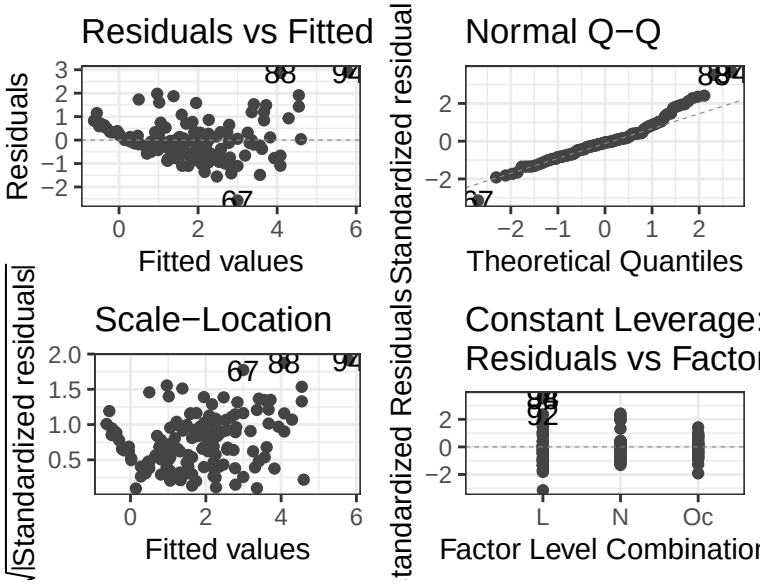
e)

```r
autoplot(fit.lm)  # Residual analysis on our regression model
```



The Residuals vs. fitted plot, or Tukey-Anscombe, seems to be nonlinear, which is how we want the residuals to behave. We have some outliers in the QQ-plot, especially points 7, 96 and 2. However, the majority of the points are making up a fairly straight line, so some outliers does not imply that our assumptions are violated. In the Scale-Location plot, the points seems to gather a little bit as the fitted values increase. This can be a sign of non-equal variances, which is violating our assumptions. In the Constant Leverage: Residuals vs Factor Levels plot, we can see that the spread of the points seems to differ based on the species. While there is only one obvious outlier for the Lumbricus species, there are more outliers for the two other species, even though these are not as far from the majority as the outlier in L. Also, note that the same observations from the QQ-plot seems to be the problematic ones.

```r
# Compare to the model without transformed variables
fit.lm1 = lm(GEWICHT ~ MAGENUMF + Gattung, data = d.worm)
autoplot(fit.lm1, smooth.colour = NA)
```



Comparing our model with transformed variables to the model without any transformations, we can clearly see that the latter one is violating the assumptions more. There seems to be a trend in the Tukey-Anscombe plot, and the points are not following the line quite as well in the QQ-plot. However, the Scale-Location plot does not behave as in our transformed model - here the points are fairly well distributed. We do have some outliers and problematic observations in the Leverage plot, but they are in general more gathered here than in the model we have used.

f)

It is important to carry out a residual analysis since we should verify that the underlying assumptions are not violated - if they are so, the model is not valid. In case of violated assumptions, we can for instance remove the outliers and fit the model over again, or we can try other transformations of the predictor value(s).

g)

FALSE, FALSE, FALSE, TRUE.

Problem 3

In this problem, we will use a dataset from the Wimbledon tennis tournament for Women in 2013. We will predict the result for player 1 (win=1 or loose=0) based on the number of aces won by each player and the number of unforced errors commited by both players. The data set is a subset of a data set from https://archive.ics.uci.edu/ml/datasets/Tennis+Major+Tournament+Match+Statistics, see that page for information of the source of the data.

The files can be read using the following code.

```
##         Player1          Player2       Result           ACE.1
##  S.Lisicki  : 6   K.Flipkens : 5   Min.   :0.0000   Min.   : 0.000
##  M.Bartoli  : 5   N.Li       : 5   1st Qu.:0.0000   1st Qu.: 1.000
##  A.Radwanska: 4   D.Cibulkova: 3   Median :1.0000   Median : 2.000
##  P.Kvitova  : 4   E.Makarova : 3   Mean   :0.5339   Mean   : 2.975
##  R.Vinci    : 4   F.Pennetta : 3   3rd Qu.:1.0000   3rd Qu.: 4.000
##  S.Stephens : 4   K.Kanepi   : 3   Max.   :1.0000   Max.   :14.000
##  (Other)    :91   (Other)    :96
##      UFE.1           ACE.2           UFE.2
##  Min.   : 4.00   Min.   : 0.000   Min.   : 2.00
##  1st Qu.:13.00   1st Qu.: 1.000   1st Qu.:12.00
##  Median :18.00   Median : 2.000   Median :18.00
##  Mean   :20.18   Mean   : 3.271   Mean   :20.47
##  3rd Qu.:25.75   3rd Qu.: 5.000   3rd Qu.:27.00
##  Max.   :54.00   Max.   :15.000   Max.   :55.00
##
```

a)

Probability to win for player 1 given by a logistic regression model:

$$P(Y_i = 1|\mathbf{X} = \boldsymbol{x}_i) = p_i = \frac{e^{\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}+\beta_3 x_{i3}+\beta_4 x_{i4}}}{1 + e^{\beta_0+\beta_1 x_{i1}+\beta_2 x_{i2}+\beta_3 x_{i3}+\beta_4 x_{i4}}}$$

We first define

$$\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \beta_4 x_{i4} = \beta x$$

Furthermore we know that

$$\begin{aligned} 1 - p_i &= 1 - \frac{e^{\beta x}}{1 + e^{\beta x}} \\ &= \frac{1 + e^{\beta x} - e^{\beta x}}{1 + e^{\beta x}} \\ &= \frac{1}{1 + e^{\beta x}} \end{aligned}$$

Hence we find the linear relation between p_i and the covariates:

$$\begin{aligned} \text{logit}(p_i) &= \log(\frac{p_i}{1 - p_i}) \\ &= \log(p_i) - \log(1 - p_i) \\ &= \log(e^{\beta x}) - \log(1 + e^{\beta x}) - \log(1) + \log(1 + e^{\beta x}) \\ &= \beta x - 0 \\ &= \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \beta_4 x_{i4} \end{aligned}$$

b)

We provide an interpretation of how the coefficients β_i affect p_i by looking at the relative change in odds $\frac{p_i}{1-p_i}$ when increasing a covariate β_i.

$$\begin{aligned} \frac{\text{odds}(Y_i = 1 \mid X_1 = x_{i1} + 1)}{\text{odds}(Y_i = 1 \mid X_j = x_{ij})} &= \frac{e^{\beta_0 + \beta_1 (x_{i1} + 1) + \beta_2 x_{i2} + \beta_3 x_{i3} + \beta_4 x_{i4}}}{e^{\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \beta_4 x_{i4}}} \\ &= e^{\beta_1 1} \end{aligned}$$

Hence we see that the odds will increase with a factor of β_1 if player 1 has one more ace, $x'_{i1} = x_{i1} + 1$. As $\beta_1 = 0.36338$ we know that one more ace for player 1 will increase the likelihood for player 1 winning the match with a factor of $e^{0.36338} \approx 1.438$, in our model.

```
##
## Call:
## glm(formula = Result ~ ACE.1 + ACE.2 + UFE.1 + UFE.2, family = "binomial",
##     data = tennis)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0517  -0.8454   0.3725   0.8773   2.0959
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.02438    0.59302  -0.041 0.967211
## ACE.1        0.36338    0.10136   3.585 0.000337 ***
## ACE.2       -0.22388    0.07369  -3.038 0.002381 **
## UFE.1       -0.09847    0.02840  -3.467 0.000527 ***
```

```
## UFE.2        0.09010    0.02479   3.635 0.000278 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 163.04  on 117  degrees of freedom
## Residual deviance: 124.96  on 113  degrees of freedom
## AIC: 134.96
##
## Number of Fisher Scoring iterations: 4
```

c)

We will now use a 0.5 cutoff rule, meaning we classify an observation $\boldsymbol{x}$ as a win for player 1 if $P(Y = 1|x) > 0.5$. We find the boundary from

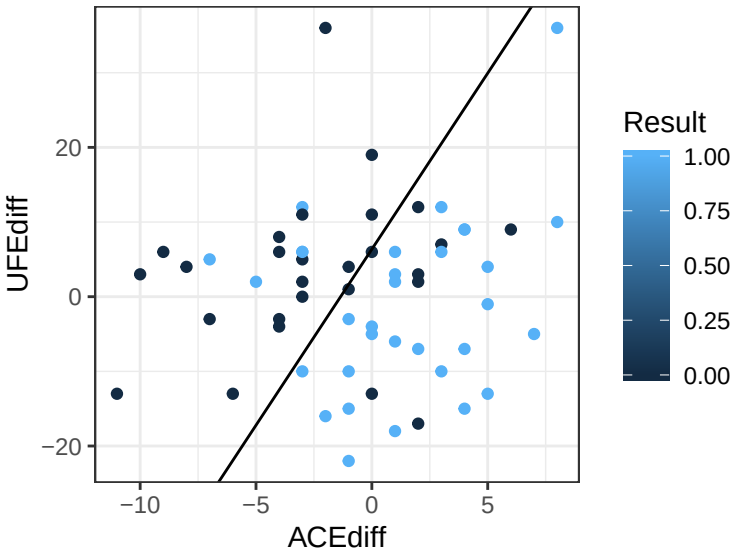
$$\begin{aligned} P(Y_i = 1|\mathbf{X} = \boldsymbol{x}_i) &= p_i \\ &= \frac{e^{\beta x}}{1 + e^{\beta x}} = 0.5 \\ &\implies e^{\beta x} = 1 \\ &\implies \beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0 \\ &\implies x_2 = -\frac{\beta_1}{\beta_2} x_1 - \frac{\beta_0}{\beta_2} \end{aligned}$$

Hence we can plot the line $y = ax + b$ with $a = -\frac{\beta_1}{\beta_2}$ and $b = -\frac{\beta_0}{\beta_2}$.

We can now provide a plot with `ACEdiff` as x-axis, `UFEdiff` as y-axis, color the points to indicate win or loss together with our computed boundary. This gives a visual understanding of our classification model.

```
# make variables for difference
tennis$ACEdiff = tennis$ACE.1 - tennis$ACE.2
tennis$UFEdiff = tennis$UFE.1 - tennis$UFE.2

# divide into test and train set
n = dim(tennis)[1]
n2 = n/2
set.seed(1111)  # to reproduce the same test and train sets each time you run the code
train = sample(c(1:n), replace = F)[1:n2]
tennisTest = tennis[-train, ]
tennisTrain = tennis[train, ]
```

```
r.tennisLogres = glm(Result ~ ACEdiff + UFEdiff, data = tennisTrain, family = "binomial")

beta = r.tennisLogres$coefficients
a = -beta[1]/beta[3]
b = -beta[2]/beta[3]

plot = ggplot(tennisTrain, aes(x = ACEdiff, y = UFEdiff, color = Result)) + geom_point() +
    geom_abline(slope = b, intercept = a) + theme_bw()

plot
```



Furthermore we can use our model to predict win or loss by player 1 on the test set. As our data set includes the row `Result` we know whether player 1 actually won or not. Hence we find the confusion matrix to measure the accuracy of our model. This also enables us to find the sensitivity and specificity. If TP are true positives, P the sum of predicted and actual positives, TN are true negatives, N the sum of predicted and actual negatives, we have sensitivity given by:

$$\frac{\text{TP}}{\text{P}}$$

And specificity by:

$$\frac{\text{TN}}{\text{N}}$$

```
##      pred
## real  0  1
##    0 16 12
##    1  5 26
```

```
## [1] "Sensitivity: 0.838709677419355"
```

```
## [1] "Specificity: 0.571428571428571"
```

d)

We will now use Linear discriminant analysis (LDA) to classify in the same manner as we did above. Using LDA we assume that our data follow known normal distributions, and we use Bayes theorem to find $P(Y|X)$. Given that we have K classes we find the posterior probability by

$$P(Y = k|\mathbf{X} = \boldsymbol{x}) = \frac{\pi_k f_k(\boldsymbol{x})}{\sum_{l=1}^{K} \pi_l f_l(\boldsymbol{x})}$$

First of all we have $\pi_k = P(Y = k)$ which is called the *prior probability*. In our case π_1 is the probability for player 1 winning any match, and π_0 is the probability for loss. Given a data set, as we have from `tennis`, we estimate by dividing total number of wins n_w by total number of losses n_l by player 1:

$$\hat{\pi} = \frac{n_w}{n_l}$$

μ_k is the expectation of the covariate x for the different classes. In our case x is a multivariate random vector with two random variables $x = [x_1, x_2]^T$. x_1 represents `ACEdiff` and x_2 `UFEdiff`. μ_1 is the expecation of x given $Y = 1$, meaning player 1 won the match, $\mathrm{E}[x|Y = 1]$, in the same way μ_0 is the expecation of x in the cases where player 1 lost the match, $\mathrm{E}[x|Y = 0]$. μ_k is estimated by

$$\hat{\boldsymbol{\mu}}_k = \frac{1}{n_k} \sum_{i:y_i=k} \mathbf{X}_i$$

Σ is the covariance matrix for x, or for `ACEdiff` and `UFEdiff`, our random variables in this case. With a given data set we have the covariance matrix for class k:

$$\hat{\boldsymbol{\Sigma}}_k = \frac{1}{n_k - 1} \sum_{i:y_i=k} (\mathbf{X}_i - \hat{\boldsymbol{\mu}}_k)(\mathbf{X}_i - \hat{\boldsymbol{\mu}}_k)^T$$

Which we use to find the use the following estimator for the covariance

$$\hat{\boldsymbol{\Sigma}} = \sum_{k=1}^{K} \frac{n_k - 1}{n - K} \cdot \hat{\boldsymbol{\Sigma}}_k$$

e)

Expression

We will now derive an expression for $\delta_0(\boldsymbol{x}) = \delta_1(\boldsymbol{x})$ starting with

$$P(Y = 0|\mathbf{X} = \boldsymbol{x}) = P(Y = 1|\mathbf{X} = \boldsymbol{x})$$

$$\frac{\pi_0 f_0(\boldsymbol{x})}{\pi_0 f_0(\boldsymbol{x}) + \pi_1 f_1(\boldsymbol{x})} = \frac{\pi_1 f_1(\boldsymbol{x})}{\pi_0 f_0(\boldsymbol{x}) + \pi_1 f_1(\boldsymbol{x})}$$

$$\pi_0 f_0(\boldsymbol{x}) = \pi_1 f_1(\boldsymbol{x})$$

Where $f_k(\boldsymbol{x})$ is defined as

$$f_k(\boldsymbol{x}) = \frac{1}{(2\pi)^{p/2}|\boldsymbol{\Sigma}|^{1/2}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu_k})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu_k})}$$

We see that the term outside the exponent cancels because of the assumption that $\Sigma_0 = \Sigma_1 = \Sigma$, hence we take log on both sides to get:

$$-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu_0})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu_0}) + \log \pi_0 = -\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu_1})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu_1}) + \log \pi_1$$

Finally terms independent of k cancels, like $x^T \Sigma^{-1} x$. Furthermore, by the symmetry of Σ and the fact that $b^T = b$ when $b \in \mathbb{R}$ we get

$$\boldsymbol{x}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_0 - \frac{1}{2}\boldsymbol{\mu}_0^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_0 + \log\pi_0 = \boldsymbol{x}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 - \frac{1}{2}\boldsymbol{\mu}_1^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 + \log\pi_1$$
$$\delta_0(\boldsymbol{x}) = \delta_1(\boldsymbol{x})$$

Function for class boundary

We will now derive the class boundary by using the expression we have derived above. As we know that

$$P(Y = 0|\mathbf{X} = \boldsymbol{x}) = P(Y = 1|\mathbf{X} = \boldsymbol{x}) = 0.5$$

We can use $\delta_0(\boldsymbol{x}) = \delta_1(\boldsymbol{x})$ to find

$$\boldsymbol{x}^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1) - \frac{1}{2}\boldsymbol{\mu}_0^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_0 + \frac{1}{2}\boldsymbol{\mu}_1^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 + \log\frac{\pi_0}{\pi_1} = 0$$

By defining

$$[\alpha_1, \alpha_2] = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)$$

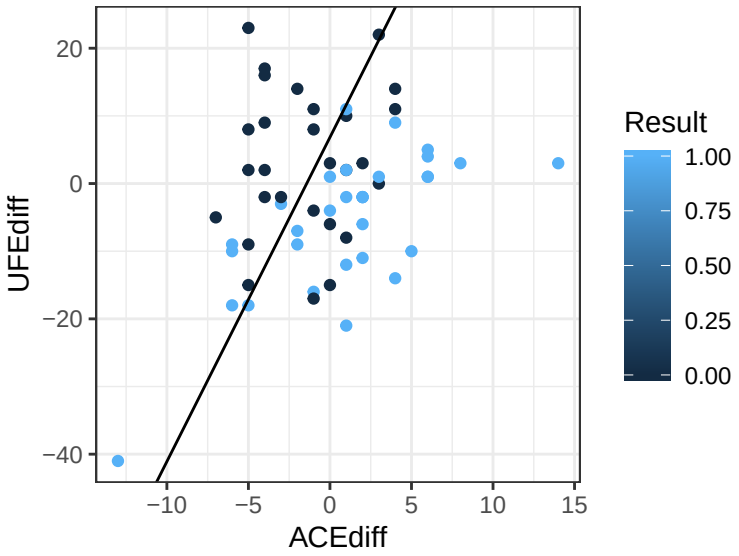
we get

$$[x_1, x_2]^T[\alpha_1, \alpha_2] = x_1\alpha_1 + x_2\alpha_2$$

and finally we have

$$x_2 = x_1\frac{\alpha_1}{\alpha_2} + \frac{1}{2\alpha_2}(\boldsymbol{\mu}_0^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 - 2\log\frac{\pi_0}{\pi_1})$$

```
n = nrow(tennisTrain)

r1 = tennisTrain[tennisTrain$Result == 1, ]
r1 = select(r1, ACEdiff, UFEdiff)
mu1 = colMeans(r1)
cov1 = cov(r1)
pi1 = nrow(r1)/n

r0 = tennisTrain[tennisTrain$Result == 0, ]
r0 = select(r0, ACEdiff, UFEdiff)
mu0 = colMeans(r0)
cov0 = cov(r0)
pi0 = nrow(r0)/n

covEst = (1/(n - 2)) * ((nrow(r0) - 1) * cov0 + ((nrow(r1) - 1) * cov1))
covInv = solve(covEst)

alpha = covInv %*% (mu0 - mu1)
a = -alpha[1]/alpha[2]
b = (1/(2 * alpha[2])) * (t(mu0) %*% covInv %*% mu0 - t(mu1) %*% covInv %*% mu1 -
    2 * log(pi0/pi1))

plot = ggplot(tennisTest, aes(x = ACEdiff, y = UFEdiff, color = Result)) + geom_point() +
    geom_abline(slope = a, intercept = b) + theme_bw()

plot
```



f)

```
r.tennisLDA = lda(Result ~ ACEdiff + UFEdiff, data = tennisTrain)
tennisLDAPred = predict(r.tennisLDA, tennisTest)$class

tennisLDACM = table(real = tennisTest$Result, predicted = tennisLDAPred)
tennisLDACM
```

```
##     predicted
## real  0  1
##    0 16 12
##    1  5 26
```

```
sensLDA = tennisLDACM[2, 2]/sum(tennisLDACM[2, ])
print(paste0("Sensitivity: ", sensLDA))
```

```
## [1] "Sensitivity: 0.838709677419355"
```

```
specLDA = tennisLDACM[1, 1]/sum(tennisLDACM[1, ])
print(paste0("Specificity: ", specLDA))
```

```
## [1] "Specificity: 0.571428571428571"
```

g)

When we derived the decision boundary for LDA we assumed that each class have equal covariance matrices. For QDA we do not assume this and hence the quadratic terms of x do not cancel and we obviously get a decision boundary that is quadratic.

```
r.tennisQDA = qda(Result ~ ACEdiff + UFEdiff, data = tennisTrain)
tennisQDAPred = predict(r.tennisQDA, tennisTest)$class

tennisQDACM = table(real = tennisTest$Result, predicted = tennisQDAPred)
tennisQDACM
```

```
##     predicted
## real  0  1
##    0 15 13
##    1  5 26
```

```
sensQDA = tennisQDACM[2, 2]/sum(tennisQDACM[2, ])
print(paste0("Sensitivity: ", sensQDA))
```

```
## [1] "Sensitivity: 0.838709677419355"
```

```
specQDA = tennisQDACM[1, 1]/sum(tennisQDACM[1, ])
print(paste0("Specificity: ", specQDA))
```

```
## [1] "Specificity: 0.535714285714286"
```

h)

For one run with seed = 1234 we get the following sensitivity and specificity for the different methods.

	Sens	Spec	SUM
glm	0.80	0.76	1.56
LDA	0.83	0.69	1.52
QDA	0.80	0.69	1.49

However, in order to get a result that is not dependent on only one sample, we can run the classifiers n times with new random test / training samples each time, and then taking the average of the sensitivity and specificity for each method. Here is the result for running the function below for $n = 1000$:

	Sens	Spec	SUM
glm	0.78	0.68	1.46
LDA	0.79	0.67	1.46
QDA	0.79	0.65	1.43

First of all, the performance differences between the methods are not that great, but we see that QDA has a somewhat weaker specificity, which means it is weaker at separating the "negatives", meaning the cases where player 1 lost the match. This could imply that the best model for `ACEdiff` and `UFEdiff` is linear, not quadratic as shown in Figure 3. It also seems reasonable that the descision boundary should increase on the whole domain, but in Figure 3 we observe a decreasing slope for ACEdiff < -5. This means that when player 2 has a relative lead of 5 aces or more, player 1 is suddenly "allowed" to have a relative larger number of unforced errors, for the match to be a likely win for player 1. This does not make intuitive sense.

Hence we would prefer to use glm or LDA. However, as LDA has a slightly better probability of classifying

the matches where player 1 won, it could be preferable if we care more about predicting wins than losses.

```
tennis_classification = function(runs) {
    res = matrix(0L, nrow = 3, ncol = 2)

    for (i in 1:runs) {

        # make variables for difference
        tennis$ACEdiff = tennis$ACE.1 - tennis$ACE.2
        tennis$UFEdiff = tennis$UFE.1 - tennis$UFE.2

        # divide into test and train set
        n = dim(tennis)[1]
        n2 = n/2
        train = sample(c(1:n), replace = F)[1:n2]
        tennisTest = tennis[-train, ]
        tennisTrain = tennis[train, ]

        # Logistical regression
        r.tennisLogres = glm(Result ~ ACEdiff + UFEdiff, data = tennisTrain, family = "binomial")
        tennisLogresPred = predict(r.tennisLogres, newdata = tennisTest, type = "response")
        tennisLogresResult = ifelse(tennisLogresPred < 0.5, 0, 1)
        tennisLogresCM = table(real = tennisTest$Result, pred = tennisLogresResult)
        res[1, 1] = res[1, 1] + tennisLogresCM[2, 2]/sum(tennisLogresCM[2, ])
        res[1, 2] = res[1, 2] + tennisLogresCM[1, 1]/sum(tennisLogresCM[1, ])

        # LDA
        r.tennisLDA = lda(Result ~ ACEdiff + UFEdiff, data = tennisTrain)
        tennisLDAPred = predict(r.tennisLDA, tennisTest)$class
        tennisLDACM = table(real = tennisTest$Result, predicted = tennisLDAPred)
        res[2, 1] = res[2, 1] + tennisLDACM[2, 2]/sum(tennisLDACM[2, ])
        res[2, 2] = res[2, 2] + tennisLDACM[1, 1]/sum(tennisLDACM[1, ])

        # QDA
        r.tennisQDA = qda(Result ~ ACEdiff + UFEdiff, data = tennisTrain)
        tennisQDAPred = predict(r.tennisQDA, tennisTest)$class
        tennisQDACM = table(real = tennisTest$Result, predicted = tennisQDAPred)
        res[3, 1] = res[3, 1] + tennisQDACM[2, 2]/sum(tennisQDACM[2, ])
        res[3, 2] = res[3, 2] + tennisQDACM[1, 1]/sum(tennisQDACM[1, ])
    }

    return(res/runs)
}

res = tennis_classification(1000)
res
```

```
##           [,1]      [,2]
## [1,] 0.7801013 0.6775830
## [2,] 0.7871556 0.6713058
## [3,] 0.7818750 0.6456521
```

Problem 4

a)

10 fold cross validation on the KNN regression would be performed by partitioning the training data D into 10 equal size sets D_i. For each of the 10 sets, leave the set out form the data and calculate a test error using the set as test data. Then average the test errors for all the 10 sets. More precisely we would use test MSE as error measure. Let $\mathcal{N}_i(x)$ be the K closest points in $D \setminus D_i$ to x. The test MSE is calculated as

$$\text{MSE}_i = \frac{1}{|D_i|}\sum_{j=1}^{|D_i|}(y_j - \frac{1}{K}\sum_{l \in \mathcal{N}_i(x_j)} y_l)^2,$$

where $y_j, y_j \in D_i$ and y_l is the observed response at l. The validation error is then calculated as

$$\text{CV}_{10} = \frac{1}{10}\sum_{i=10}^{10}\text{MSE}_i.$$

b)

TRUE, TRUE, TRUE, FALSE

c)

```
id <- "1I6dk1fA4ujBjZPo3Xj8pIfnzIa94WKcy"  # google file ID
d_chd <- read.csv(sprintf("https://docs.google.com/uc?id=%s&export=download", id))
get_f_hat <- function(chd_data) {
    logitRegChd <- glm(chd ~ sbp + sex, data = chd_data, family = "binomial")
    b <- coef(logitRegChd)[1]
    bSpb <- coef(logitRegChd)[2]
    bSex <- coef(logitRegChd)[3]
    return(function(spb, sex) {
        return(exp(b + bSpb * spb + bSex * sex)/(1 + exp(b + bSpb * spb + bSex *
            sex)))
    })

}
f_hat <- get_f_hat(d_chd)
chdProbability <- unname(f_hat(140, 1))

# testPlotFemale <- ggplot(subset(d_chd, sex == 0), aes(x=subset(sbp, sex == 0),
# y=subset(chd, sex == 0)))+geom_point() + geom_line(aes(x=sbp, y=f_hat(sbp,1)),
# col='blue') testPlotMale <- ggplot(subset(d_chd, sex == 1), aes(x=subset(sbp,
# sex == 1), y=subset(chd, sex == 1)))+geom_point() + geom_line(aes(x=sbp,
# y=f_hat(sbp,2)), col='blue') testPlotFemale testPlotMale
```

The probability of coronary heart disease for a male with a blood pressure 140 is 0.3831131.

d)

```
eval_f_hat <- function(sbp, sex) {
    return(function(data, indexs) {
        get_f_hat(data[indexs, ])(sbp, sex)
    })
}
boostrap_stats <- function(data, stat_func, num_samples) {
    rows <- nrow(data)
    boostrap_stats <- numeric(num_samples)
    for (b in 1:num_samples) {
        indexs = sample.int(n = rows, size = rows, replace = TRUE)
        boostrap_stats[b] <- stat_func(data, indexs)
    }
    return(boostrap_stats)
}
probs <- boostrap_stats(d_chd, eval_f_hat(140, 1), 1000)
# bootProbs<- boot(data = d_chd,statistic = eval_f_hat(140,1) , R = 1000)
stdError = sd(probs)
# ggplot(data = data.frame(x=bootProbs$t),aes(x=x))+ geom_density() +
# geom_density(data = data.frame(x=probs),aes(x=x), color = 'red') Confidence
# interval Removing top 1000*(1-a/2) from each end
low = Rfast::nth(probs, 26, descending = FALSE)
high = Rfast::nth(probs, 26, descending = TRUE)
```

From the probabilies we get a standard error of 0.0474945. A 95% confidence interval for the the estimator of $P(\text{chd}|\text{sex} = \text{male}, \text{sbp} = 40)$ is [0.2946268, 0.4859706]. If we assume the estimated distribution to be true it means that the estimator will take values in [0.2946268 ,0.4859706] 95% of the time. That is, we estimate that the there is a 95% chance that males with blood pressure 140 has between 0.2946268 to 0.4859706 probability of having coronary heart disease. This is quite high, and indicates that the estimator has a high variance, but we don't know if it has low or high bias.