TMA4315: Compulsory exercise 2

Group 0: Jakob Bergset Heide

13.10.2023

In the first part of this exercise, we will consider a logistic regression model on a grouped data set. In the second part, we will fit a Poisson regression model on another dataset, namely matches played in "Eliteserien", and try to predict the winner of this year's league.

Part 1 - Logistic regression

a)

We start by deriving the log-likelihood function of the logistic regression. By definition, the likelihood function is

$$L(\beta) = \prod_{i=1}^{113} f(y_i;\beta) = \prod_{i=1}^{113} \binom{n_i}{y_i} \pi_i^{y_i}(1-\pi_i)^{n_i-y_i}$$

Taking the log, we get the log-likelihood function

$$\begin{aligned} l(\beta) = \ln L(\beta) &= \sum_{i=1}^{113} \ln\left(\binom{n_i}{y_i} \pi_i^{y_i}(1-\pi_i)^{n_i-y_i}\right) \\ &= \sum_{i=1}^{113}\left(\ln\binom{n_i}{y_i} + y_i\ln(\pi_i) + (n_i-y_i)\ln(1-\pi_i)\right) \end{aligned}$$

Inserting $\pi_i = \frac{e^{\eta_i}}{1+e^{\eta_i}}$ and $\eta_i = x_i^T\beta$, we get

$$\begin{aligned} l(\beta; x_i, y_i) &= \sum_{i=1}^{113}\left(\ln\binom{n_i}{y_i} + y_i\ln\Big(\frac{e^{\eta_i}}{1+e^{\eta_i}}\Big) + (n_i-y_i)\ln\Big(1-\frac{e^{\eta_i}}{1+e^{\eta_i}}\Big)\right) \\ &= \sum_{i=1}^{113}\left(\ln\binom{n_i}{y_i} + y_i\ln(e^{\eta_i}) + n_i\ln\left(\frac{1}{1+e^{\eta_i}}\right)\right) \\ &= \sum_{i=1}^{113}\left(\ln\binom{n_i}{y_i} + y_i x_i^T\beta - n_i\ln(1+e^{x_i^T\beta})\right) \end{aligned}$$

In order to find maximum likelihood estimates, we would attempt to solve a set of equations, namely

$$\frac{\partial l(\beta)}{\partial \beta} = \mathbf{0}$$

where the left hand side is commonly known as the score vector. There are p non-linear equations to solve (in our case, $p = 2$), and so we would need to employ some numerical optimization scheme. Since the log-likelihood is a concave function, a natural choice would be Newton's method (or some variation of it), since it gains quadratic convergence when applied to a convex function.

b)

```r
filepath <- "https://www.math.ntnu.no/emner/TMA4315/2018h/mountains"
mount <- read.table(file = filepath, header = TRUE, col.names = c("height",
    "prominence", "fail", "success"))

logistic.fit <- glm(cbind(success, fail) ~ height + prominence, data = mount,
    family = "binomial")
logistic.fit
```

```
##
## Call:  glm(formula = cbind(success, fail) ~ height + prominence, family = "binomial",
##     data = mount)
##
## Coefficients:
## (Intercept)       height   prominence
##   13.685845    -0.001635    -0.000174
##
## Degrees of Freedom: 112 Total (i.e. Null);  110 Residual
## Null Deviance:       715.3
## Residual Deviance: 414.7     AIC: 686
```

We can interpret the coefficient estimates as follows:

- the coefficient for `height` is -0.001635, which indicates that if we increase `height` by 1 (and keep other covariates fixed), then the odds for success is multiplied by $e^{-0.001635} = 0.998366$, i.e. the odds for success decrease.
- similarly, the coefficient for `prominence` is -0.000174, so if we increase `prominence` by 1, the odds for success is multiplied by $e^{-0.000174} = 0.999826$, i.e. the odds for success decrease.

We want to test the significance of our coefficient estimates. That is, we want to test the hypotheses

$$H_0 : \beta_{height} = 0$$
$$H_0 : \beta_{prominence} = 0$$

We know the asymptotic distribution of our parameters: $\hat{\beta} \sim N_p(\beta, F^{-1}(\hat{\beta}))$. F is the expected Fisher information matrix, which we could calculate as $F(\beta) = \mathbf{X}^T\mathbf{W}\mathbf{X}$, where $\mathbf{W} = \text{diag}(n_i\pi_i(1-\pi_i))$. For a general hypothesis test $H_0 : \mathbf{C}\beta = \mathbf{d}$, the Wald test uses the test statistic

$$w = (\mathbf{C}\hat{\beta} - \mathbf{d})^T(\mathbf{C}F^{-1}(\hat{\beta})\mathbf{C}^T)^{-1}(\mathbf{C}\hat{\beta} - \mathbf{d}) \sim \chi^2_r$$

which is asymptotically χ^2 with r (number of parameters tested) degrees of freedom. In our case, with only one parameter in each test, the test statistic becomes

$$w = \frac{(\hat{\beta}_j - 0)^2}{\text{Var}(\hat{\beta}_j)} \sim \chi^2_1$$

which we can use to calculate a p-value and report statistical significance. We extract the Fisher information matrix from the model fit, calculate the test statistic, and perform the hypothesis tests.

```
# Extract the inverse Fisher matrix
Fisher_inv <- summary(logistic.fit)$cov.unscaled
Fisher_inv
```

```
##               (Intercept)        height    prominence
## (Intercept)  1.132415e+00 -1.504678e-04  2.195593e-05
## height      -1.504678e-04  2.014927e-08 -3.365130e-09
## prominence   2.195593e-05 -3.365130e-09  2.073457e-09
```

```
# Calculate test statistics
test_height <- (logistic.fit$coefficient[2])^2/(Fisher_inv[2, 2])
test_prominence <- (logistic.fit$coefficient[3])^2/(Fisher_inv[3, 3])

# Calculate p-values, report significance
p_value_height <- pchisq(test_height, df = 1, lower.tail = FALSE)
p_value_prominence <- pchisq(test_prominence, df = 1, lower.tail = FALSE)

p_value_height
```

```
##       height
## 1.031229e-30
```

```
p_value_prominence
```

```
## prominence
## 0.00013302
```

We see that the p-values in the chi square test are $1.03 \cdot 10^{-30}$ and $1.33 \cdot 10^{-4}$ for `height` and `prominence`, respectively. This means that we reject the null hypothesis (for the typical level $\alpha = 0.05$) and both coefficients are deemed significant.

To create a 95% confidence interval, we note that

$$\sqrt{w} = \frac{\hat{\beta}_j - \beta}{\sqrt{\mathrm{Var}(\hat{\beta}_j)}} =: Z_j \sim N(0,1)$$

Thus, the confidence interval can be constructed from

$$P(-z_{\alpha/2} \leq Z_j \leq z_{\alpha/2}) = 1 - \alpha,$$

where $z_{\alpha/2}$ are the critical values from the standard normal distribution, given some level α. We calculate the 95% confidence interval using $\alpha = 0.05$:

```
z <- 1.96  #Critical value from normal dist.

lower_height <- logistic.fit$coefficient[2] - z * sqrt(Fisher_inv[2,
    2])
upper_height <- logistic.fit$coefficient[2] + z * sqrt(Fisher_inv[2,
    2])

cat("95% confidence interval for height coefficient: ", c(lower_height,
    upper_height), "\n")
```

```
## 95% confidence interval for height coefficient:  -0.001913636 -0.001357199
```

The 95% confidence interval for the height coefficient is $(-0.001913636, -0.001357199)$. We know that if we increase the covariate `height` by 1, then the odds for success are multiplied by $e^{\beta_{height}}$. So we can calculate the exponentials of the lower and upper limit,

```r
# Confidence intervals with exponentials
c(exp(lower_height), exp(upper_height))
```

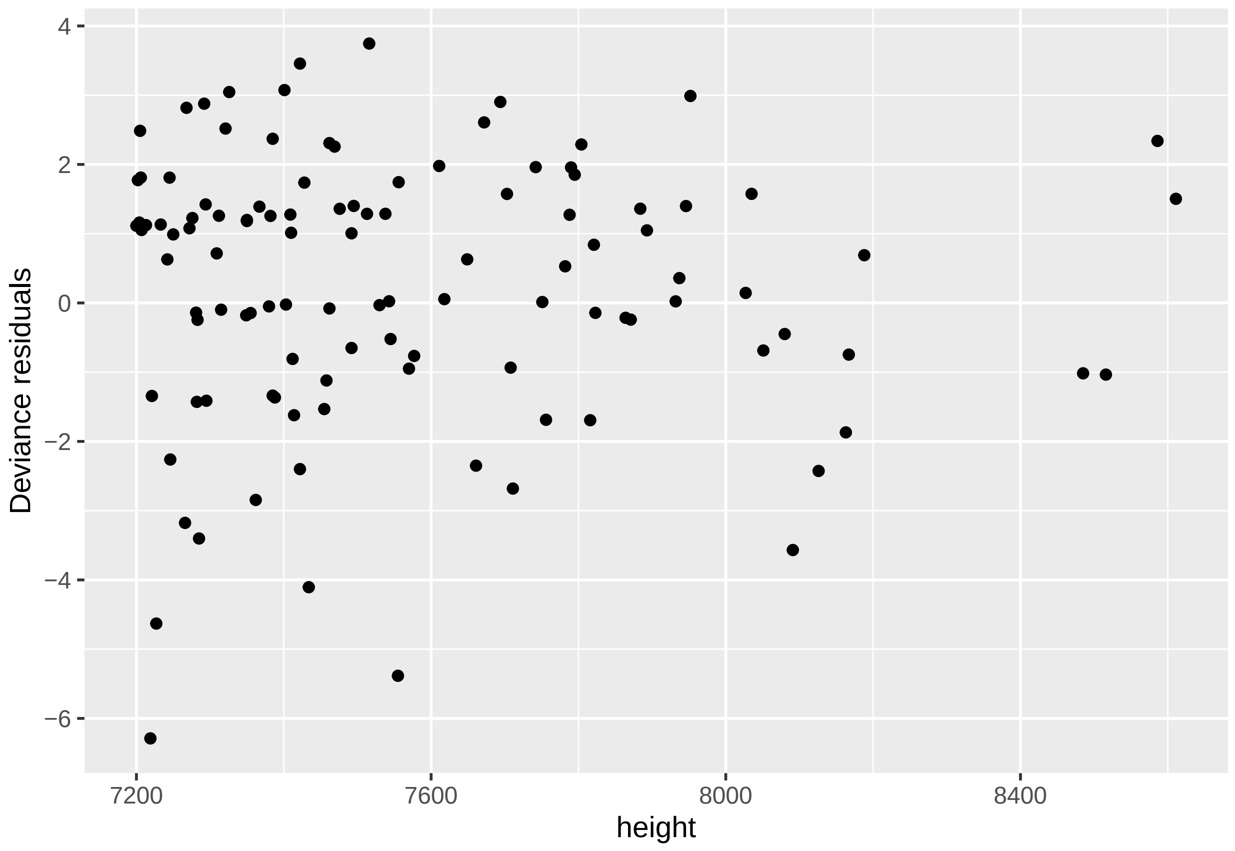
```
##    height    height
## 0.9980882 0.9986437
```

which we can interpret as a 95% confidence interval for how much the odds of success will be multiplied by, when we increase `height` by 1.

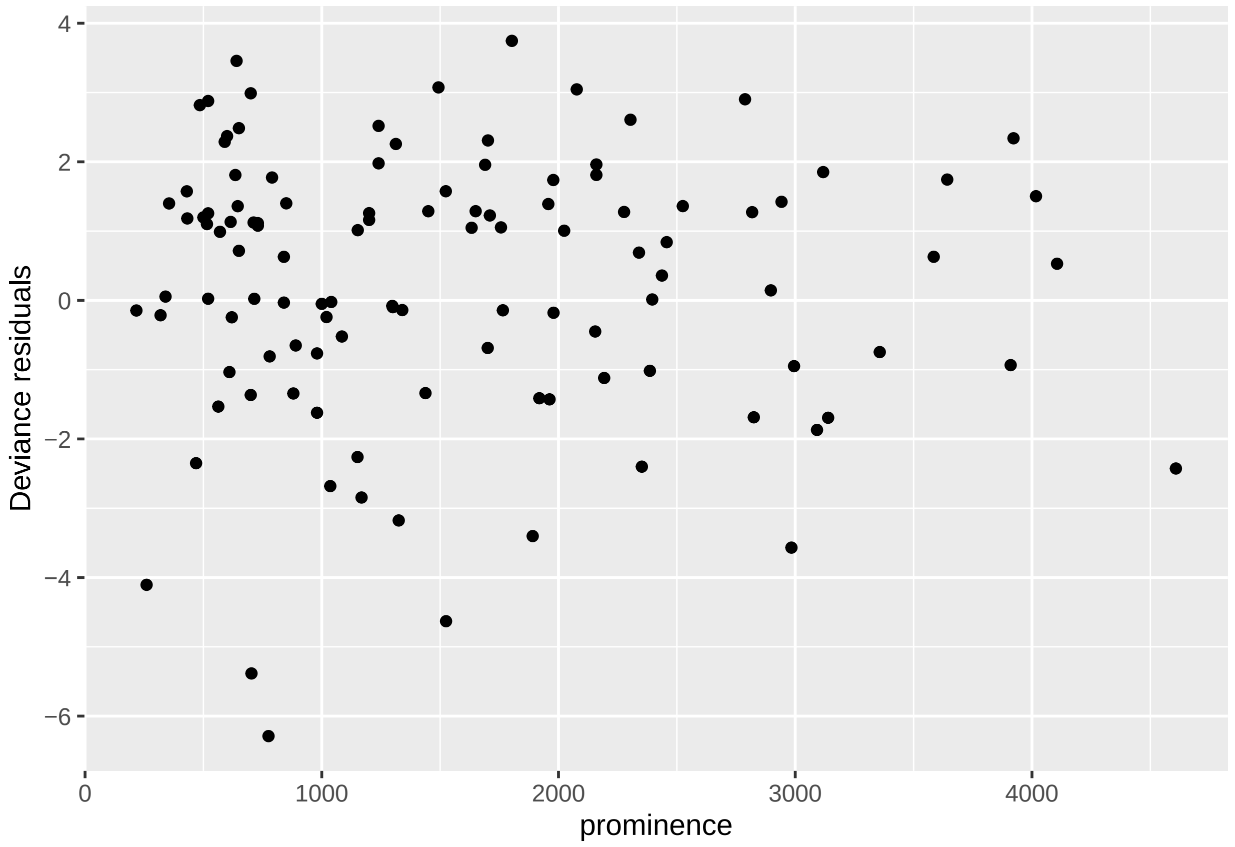
c)

To assess our model fit, we can create plots of the deviance residuals vs the covariates:

```r
# Plotting deviance residuals
library(ggplot2)
ggplot(mount, aes(x = height, y = residuals(logistic.fit, type = "deviance"))) +
    geom_point() + ylab("Deviance residuals")
```



```r
ggplot(mount, aes(x = prominence, y = residuals(logistic.fit, type = "deviance"))) +
    geom_point() + ylab("Deviance residuals")
```



We see that for both the `height` and `prominence` plots, the residuals seem to be mostly evenly distributed around the x-axis, with no apparent trends. There might be some more spread in the residuals towards the negative y-axis, which could indicate some non-constant variance in the data.

The deviance is defined as

$$D = -2(\ln L(candidate\ model) - \ln L(saturated\ model))$$

where the candidate model is our model, and the saturated model is the model where each group is given its own parameter. We can find the deviance of the model in the output of `summary(logistic.fit)` as "Residual deviance", which we can use to check the hypothesis that our model fits the data well (i.e. candidate model is not far from the saturated model). In addition, the null deviance is reported in `summary(logistic.fit)`. The null deviance is the deviance where the candidate model is the model containing only the intercept. The deviance is χ^2_{G-p} distributed and so we can perform a hypothesis test:

```r
summary(logistic.fit)
```

```
##
## Call:
## glm(formula = cbind(success, fail) ~ height + prominence, family = "binomial",
##     data = mount)
##
## Deviance Residuals:
```

```
##     Min       1Q   Median       3Q      Max
## -6.2886  -0.8086   0.6893   1.4226   3.7456
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.369e+01  1.064e+00  12.861  < 2e-16 ***
## height      -1.635e-03  1.420e-04 -11.521  < 2e-16 ***
## prominence  -1.740e-04  4.554e-05  -3.821 0.000133 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 715.29  on 112  degrees of freedom
## Residual deviance: 414.68  on 110  degrees of freedom
## AIC: 686.03
##
## Number of Fisher Scoring iterations: 4
```

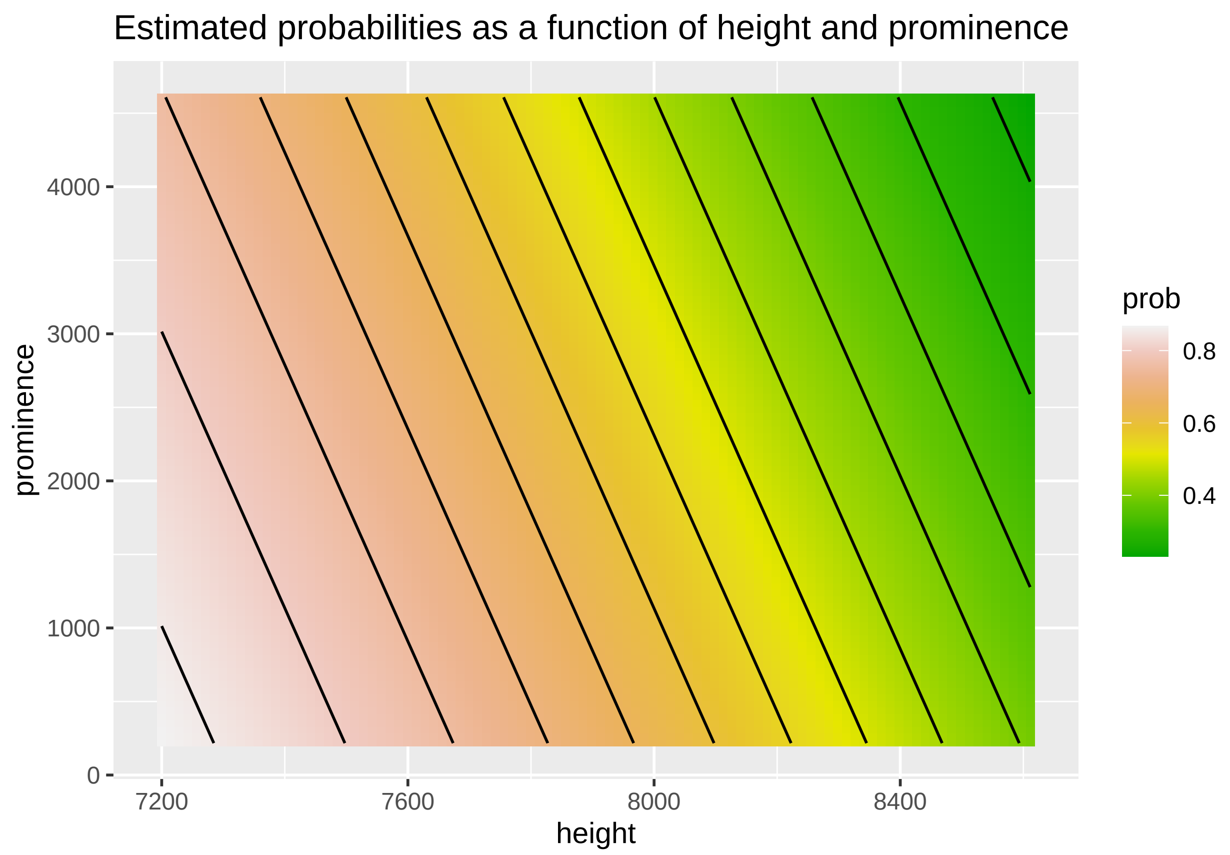
```
# Residual deviance is reported as 414.68 on 110 df
pchisq(414.68, 110, lower.tail = FALSE)
```

```
## [1] 6.622159e-37
```

With a p-value of $6.62 \cdot 10^{-37}$, we reject the null hypothesis, and conclude that our model is not a good fit to the data.

We create a heatmap of the estimated probabilites as a function of our two covariates:

```
# Plotting estimated probabilities
estimate_probability <- function(cov1, cov2) {
    coef <- logistic.fit$coefficient  #Betas for intercept, height, prominence
    eta <- c(1, cov1, cov2) %*% coef
    prob <- plogis(eta)
    return(prob)
}

n <- 100  #Number of points to evaluate probability in

# covariate vectors(x and y axis)
cov1 <- seq(min(mount$height), max(mount$height), length.out = n)
cov2 <- seq(min(mount$prominence), max(mount$prominence), length.out = n)
prob_frame <- data.frame()

Z <- matrix(NA, nrow = n, ncol = n)
for (i in 1:n) {
    for (j in 1:n) {
        Z[i, j] = estimate_probability(cov1[j], cov2[i])
        prob_frame <- rbind(prob_frame, c(cov1[j], cov2[i], Z[i, j]))
    }
}
names(prob_frame) <- c("cov1", "cov2", "prob")

ggplot(prob_frame, mapping = aes(x = cov1, y = cov2, z = prob)) + geom_raster(aes(fill = prob)) +
    geom_contour(col = "black") + scale_fill_gradientn(colours = terrain.colors(10)) +
    xlab("height") + ylab("prominence") + ggtitle("Estimated probabilities as a function of height and p
```



We see that the probability of ascending mountains decrease with increasing height and prominence. The black level lines indicate constant probability. From the plot, it is clear that height is the more influential predictor, since the level lines are very steep.

d)

We consider Mount Everest, which has both height and prominence of 8848 meters. Based on our fitted model, we can estimate the probability of successfully ascending it:

```
# Finding the probability of ascending Mount Everest
prob_everest <- estimate_probability(8848, 8848)
cat("The estimated probability of ascending Mount Everest is: ", prob_everest,
    "\n")
```

```
## The estimated probability of ascending Mount Everest is:  0.08917783
```

The estimated probability of ascending Mount Everest, based on our model, is approximately 0.0892. From before, we know that $\hat{\beta} \sim N_p(\beta, F^{-1}(\hat{\beta}))$. Since $x^T\hat{\beta}$ is a linear combination of a p-variate normal vector, we know that $x^T\hat{\beta}$ is a univariate normal with mean

$$E[x^T\hat{\beta}] = x^T\beta$$

and variance

$$\text{Var}(x^T\hat{\beta}) = x^T\text{Var}(\hat{\beta})x = x^T F^{-1}(\hat{\beta})x$$

Therefore,

$$Z := \frac{x^T\hat{\beta} - x^T\beta}{(x^T F^{-1}(\hat{\beta})x)^{1/2}} \sim N(0,1)$$

from which we can construct the $(1-\alpha)\%$ confidence interval

$$P(-z_{\alpha/2} \leq Z \leq z_{\alpha/2}) = 1-\alpha$$
$$\Rightarrow P(x^T\hat{\beta} - z_{\alpha/2}(x^T F^{-1}(\hat{\beta})x)^{1/2} \leq x^T\beta \leq x^T\hat{\beta} + z_{\alpha/2}(x^T F^{-1}(\hat{\beta})x)^{1/2}) = 1-\alpha$$

We can then apply the monotonic logit transform on the lower and upper limit and get a $(1-\alpha)\%$ CI for the probability estimate.

```r
z <- 1.96
x_everest <- c(1, 8848, 8848)
coef <- logistic.fit$coefficients

lower_lim <- x_everest %*% coef - z * sqrt(t(x_everest) %*% Fisher_inv %*%
    x_everest)
upper_lim <- x_everest %*% coef + z * sqrt(t(x_everest) %*% Fisher_inv %*%
    x_everest)
lower_lim <- plogis(lower_lim)
upper_lim <- plogis(upper_lim)

cat("95% CI for the estimated probability, ", prob_everest, ", is ",
    c(lower_lim, upper_lim), "\n")
```

```
## 95% CI for the estimated probability,  0.08917783 , is  0.05486523 0.1417315
```

We see that the 95% CI is $(0.05486523, 0.1417315)$. The limits are rather far apart, which indicates that the probability estimate is not very precise. In our data, Mount Everest is an outlier, since it has a prominence of 8848, which is much higher than for any other data point. It is therefore not very reasonable to use data and model to predict on Mount Everest.

We repeat the procedure with Mount Chogolisa, which has height 7665 and prominence 1624.

```r
prob_chogolisa <- estimate_probability(7665, 1624)
cat("The estimated probability of ascending Mount Chogolisa is: ", prob_chogolisa,
    "\n")
```

```
## The estimated probability of ascending Mount Chogolisa is:  0.7042924
```

```r
x_chogolisa <- c(1, 7665, 1624)
coef <- logistic.fit$coefficients

lower_lim <- x_chogolisa %*% coef - z * sqrt(t(x_chogolisa) %*% Fisher_inv %*%
    x_chogolisa)
upper_lim <- x_chogolisa %*% coef + z * sqrt(t(x_chogolisa) %*% Fisher_inv %*%
    x_chogolisa)
lower_lim <- plogis(lower_lim)
upper_lim <- plogis(upper_lim)

cat("95% CI for the estimated probability, ", prob_chogolisa, ", is ",
    c(lower_lim, upper_lim), "\n")
```

```
## 95% CI for the estimated probability,  0.7042924 , is  0.6832251 0.7245235
```

We see that the confidence interval is more narrow than in the case of Mount Everest, with values $(0.6832251, 0.7245235)$. In our data, there were 2 failures and 20 successes for Mount Chogolisa, which indicates a probability of $\frac{20}{20+2} = 0.91$. Our estimated probability of 0.7042924 is below this.

Part 2

We import our dataset:

```
filepath <- "https://www.math.ntnu.no/emner/TMA4315/2023h/eliteserien2023.csv"
eliteserie <- read.csv(file = filepath)

NGames <- table(c(eliteserie$home[!is.na(eliteserie$yh)], eliteserie$away[!is.na(eliteserie$yh)]))
RangeofGames <- range(NGames)
```

The dataset contains the following columns:

- `home`: the name of the home team.
- `away`: the name of the away team.
- `yh`: goals scored by the home team.
- `ya`: goals scored by the away team.

There is a total of 240 matches in the dataset, but only 179 matches has been played.

a)

Our modelling depends on the assumption that the number of goals scored by the home team and the away team are independent, and so we want to check if this is a reasonable assumption to make. To do this, we apply Pearson's chi-square test for independence, which uses contingency tables. We create the contingency table for our data, which lists all possible categories of our two covariates `yh` and `ya`:

```
contingency <- table(eliteserie[, 4:5])
contingency
```

```
##    ya
## yh   0  1  2  3  4  5
##   0 12 13  9  4  2  0
##   1 15 15 12  7  0  0
##   2  9 15  6  5  0  1
##   3  7 13  9  0  2  0
##   4  6  3  2  2  1  0
##   5  1  5  1  0  0  0
##   6  0  1  0  0  0  0
##   7  0  0  0  1  0  0
```

We are testing the hypothesis: `yh` and `ya` are independent. Under this hypothesis, we can create a table of the expected counts - using the fact that if two events are independent, then their joint probability is equal to the product of their marginal probabilities. In other words, if i, j are independent events with probabilities p_i, p_j, then $p(i, j) = p_i p_j$. In the table of expected counts, we can therefore calculate each element as Np_ip_j, where N is the total number of matches (sum of all elements in the contingency table), and $p_i = \frac{\text{number of matches with } i \text{ goals scored}}{N}$ (same for p_j). The table of expected counts in this case is

```
##    ya
## yh          0          1          2         3          4           5
##   0 11.1731844 14.5251397  8.7150838 4.2458101 1.11731844 0.223463687
##   1 13.6871508 17.7932961 10.6759777 5.2011173 1.36871508 0.273743017
##   2 10.0558659 13.0726257  7.8435754 3.8212291 1.00558659 0.201117318
##   3  8.6592179 11.2569832  6.7541899 3.2905028 0.86592179 0.173184358
##   4  3.9106145  5.0837989  3.0502793 1.4860335 0.39106145 0.078212291
##   5  1.9553073  2.5418994  1.5251397 0.7430168 0.19553073 0.039106145
##   6  0.2793296  0.3631285  0.2178771 0.1061453 0.02793296 0.005586592
##   7  0.2793296  0.3631285  0.2178771 0.1061453 0.02793296 0.005586592
```

We can now compute a test statistic which is chi square distributed, with (# of rows - 1)(# of cols - 1) degrees of freedom:

$$\sum \frac{(observed\ count - expected\ count)^2}{expected\ count} \sim \chi^2_{5 \cdot 7}$$

where the sum runs over all elements in the tables. With this observed test statistic, we can perform a chi squared test to get a p-value and determine if our hypothesis is statistically significant. We perform this test using the function `chisq.test`:

```
chi_test <- chisq.test(contingency)
chi_test
```

```
##
##  Pearson's Chi-squared test
##
## data:  contingency
## X-squared = 33.527, df = 35, p-value = 0.5393
```

We see that the reported p-value of 0.5393 is much higher than the typical significance level $\alpha = 0.05$, and the hypothesis is significant. The conclusion is that our assumption of `yh` and `ya` being significant is in fact reasonable.

b)

Based on the 179 matches played so far, we create a scoreboard.

```
# b) Table of results
create_scoreboard <- function(df) {
    all_teams <- unique(df$home)
    rankings <- data.frame()

    for (i in 1:length(all_teams)) {
        score <- 0
        goals_scored_vs_lost <- 0

        matches_home <- subset(df, home == all_teams[i])
        matches_home <- na.omit(matches_home)
        for (j in 1:length(matches_home$X)) {
            if (matches_home$yh[j] == matches_home$ya[j]) {
                # draw
                score <- score + 1
            } else if (matches_home$yh[j] > matches_home$ya[j]) {
                # win
                score <- score + 3
                goals_scored_vs_lost <- goals_scored_vs_lost + matches_home$yh[j] -
                  matches_home$ya[j]
            } else {
                # lose
                goals_scored_vs_lost <- goals_scored_vs_lost + matches_home$yh[j] -
                  matches_home$ya[j]
            }
        }

        matches_away <- subset(df, away == all_teams[i])
        matches_away <- na.omit(matches_away)
        for (j in 1:length(matches_away$X)) {
            if (matches_away$yh[j] == matches_away$ya[j]) {
                # draw
                score <- score + 1
            } else if (matches_away$yh[j] > matches_away$ya[j]) {
                # lose
                goals_scored_vs_lost <- goals_scored_vs_lost - matches_away$yh[j] +
                  matches_away$ya[j]
            } else {
                # win
                score <- score + 3
                goals_scored_vs_lost <- goals_scored_vs_lost - matches_away$yh[j] +
                  matches_away$ya[j]
            }
        }

        rankings <- rbind(rankings, c(all_teams[i], score, goals_scored_vs_lost))

    }
    names(rankings) <- c("Team", "Score", "Goal difference")
    rankings$Score <- as.numeric(rankings$Score)
    rankings$`Goal difference` <- as.numeric(rankings$`Goal difference`)

    scoreboard <- rankings[order(-rankings$Score, -rankings$`Goal difference`),
        ]
    row.names(scoreboard) <- NULL
    return(scoreboard)
}
prelim_scoreboard <- create_scoreboard(eliteserie)
library(knitr)
kable(prelim_scoreboard, align = "l")
```

Team	Score	Goal difference
Viking	51	21
Bodø/Glimt	49	28
Tromsø	48	11
Brann	42	14
Molde	41	24
Lillestrøm	36	7

Team	Score	Goal difference
Sarpsborg 08	34	5
Odd	30	-3
Rosenborg	29	-6
Strømsgodset	27	-3
HamKam	24	-15
Vålerenga	21	-8
Sandefjord Fotball	21	-9
Haugesund	21	-14
Stabæk	17	-16
Aalesund	12	-36

Note that for teams with the same amount of points, the deciding factor in ranking is the goal difference, which is the number of goals scored vs the number of goals scored against the team.

c)

We now move on to performing the Poisson regression and finding coefficient estimates. We start by constructing the design matrix $\mathbf{X}$, which is a 384×18 matrix.

```r
# c) constructing the design matrix X
eliteserie_edited <- na.omit(eliteserie)  #Remove matches not yet played

setup_design <- function(df) {
    number_of_data_points <- length(df$yh)
    number_of_covariates <- length(unique(df$home)) + 2
    X <- matrix(0, number_of_data_points * 2, number_of_covariates)
    X[, 1] <- 1
    X[1:(number_of_data_points), 2] <- 1
    colnames(X) <- c("Intercept", "Home", unique(df$home))

    # Loop through all data points
    for (i in 1:number_of_data_points) {
        # For each data point, we need to address two rows in the
        # design matrix
        home_team <- df[i, "home"]
        away_team <- df[i, "away"]
        X[i, home_team] <- 1
        X[i, away_team] <- -1

        X[i + number_of_data_points, home_team] <- -1
        X[i + number_of_data_points, away_team] <- 1
    }
    return(X)
}
```

There is an issue with the design matrix, however. The 16 team covariate columns are linearly dependent - we handle this by forcing the $\beta_{Rosenborg} = 0$, which is equivalent to removing the column `Rosenborg`. Effectively, we are using `Rosenborg` as a reference, where we view all the other team strengths relative to `Rosenborg`'s strength of 0. We then define the likelihood and score functions, which will be used in the optimization procedure where we find our maximum likelihood estimates $\hat{\beta}$. In the function `myglm`, we calculate the MLE by calling the function `optim`, where we choose BFGS as our procedure. BFGS uses the objective function and its gradient to find the optimum.

```r
X <- setup_design(eliteserie_edited)
X <- X[, -3]
y <- c(eliteserie_edited$yh, eliteserie_edited$ya)

likelihood <- function(beta) {
    result <- 0

    for (i in 1:length(y)) {
        xb <- as.numeric(X[i, ]) %*% beta
        result <- result + as.numeric(y[i]) * xb - exp(xb)
    }

    return(-result)
}

score_function <- function(beta) {
    result <- 0
    for (i in 1:length(y)) {
        result <- result + (as.numeric(y[i]) - exp(as.numeric(X[i, ]) %*%
            beta)) %*% as.numeric(X[i, ])
    }
    return(-result)
}

myglm <- function(response, covariates) {
    # Add code here to calculate coefficients, residuals, fitted
    # values, etc... Find coefficient estimates using BFGS

    initial_beta <- rep(0, length(covariates[1, ]))
    beta <- optim(initial_beta, likelihood, gr = score_function, method = "BFGS")$par

    # Rosenborg is reference:
    betas <- c(beta[1:2], 0, beta[3:17])

    return(betas)
}

poisson.fit <- myglm(y, X)

print(data.frame(corresponding = c("Intercept", "HomeAdvantage", unique(eliteserie$home)),
    coefficients = poisson.fit))
```

```
##         corresponding coefficients
## 1           Intercept   0.16644351
## 2       HomeAdvantage   0.35855766
## 3           Rosenborg   0.00000000
## 4            Aalesund  -0.40891048
## 5              HamKam  -0.12612816
## 6        Sarpsborg 08   0.16631937
## 7              Stabæk  -0.12274031
## 8              Tromsø   0.25014814
## 9               Brann   0.26202092
## 10          Lillestrøm   0.18162747
## 11              Viking   0.38768604
## 12          Bodø/Glimt   0.51028143
## 13           Haugesund  -0.06187098
## 14               Molde   0.40714373
## 15                 Odd   0.06743286
## 16 Sandefjord Fotball   0.00161208
## 17        Strømsgodset   0.03120329
## 18           Vålerenga  -0.01003438
```

```r
summary(glm(y ~ -1 + X, family = "poisson"))$coefficients
```

```
##                          Estimate Std. Error       z value     Pr(>|z|)
## XIntercept            0.166443509 0.06848994  2.430189011 0.0150909506
## XHome                 0.358557657 0.08744669  4.100299915 0.0000412615
## XAalesund            -0.408910491 0.16135331 -2.534255345 0.0112686585
## XHamKam              -0.126128161 0.16547582 -0.762215048 0.4459316665
## XSarpsborg 08         0.166319370 0.16866842  0.986072968 0.3240973074
## XStabæk              -0.122740313 0.16901232 -0.726221120 0.4677032266
## XTromsø               0.250148131 0.16513131  1.514843727 0.1298119838
## XBrann                0.262020921 0.16582346  1.580119747 0.1140794458
## XLillestrøm           0.181627464 0.16683868  1.088641244 0.2763121213
## XViking               0.387686032 0.16895698  2.294584267 0.0217569608
## XBodø/Glimt           0.510281428 0.16443406  3.103258674 0.0019140226
## XHaugesund           -0.061870990 0.16493095 -0.375132687 0.7075617882
## XMolde                0.407143724 0.16827533  2.419509341 0.0155414616
## XOdd                  0.067432854 0.16555546  0.407312777 0.6837782798
## XSandefjord Fotball   0.001612075 0.17025400  0.009468644 0.9924452279
## XStrømsgodset         0.031203288 0.17051966  0.182989380 0.8548063512
## XVålerenga           -0.010034384 0.16970764 -0.059127473 0.9528505765
```

We see that the coefficients found by `myglm` and `glm` are the same. Lastly, we rank the teams based on their strength parameter.

```r
strengths <- poisson.fit[3:18]
estimated_rankings <- data.frame(team = unique(eliteserie_edited$home),
    strength = strengths)
estimated_scoreboard <- estimated_rankings[order(-estimated_rankings$strength),
    ]
row.names(estimated_scoreboard) = NULL
estimated_scoreboard
```

```
##                  team    strength
## 1          Bodø/Glimt  0.51028143
## 2               Molde  0.40714373
## 3              Viking  0.38768604
## 4               Brann  0.26202092
## 5              Tromsø  0.25014814
## 6          Lillestrøm  0.18162747
## 7        Sarpsborg 08  0.16631937
## 8                 Odd  0.06743286
## 9        Strømsgodset  0.03120329
## 10 Sandefjord Fotball  0.00161208
## 11          Rosenborg  0.00000000
## 12          Vålerenga -0.01003438
## 13          Haugesund -0.06187098
## 14             Stabæk -0.12274031
## 15             HamKam -0.12612816
## 16           Aalesund -0.40891048
```

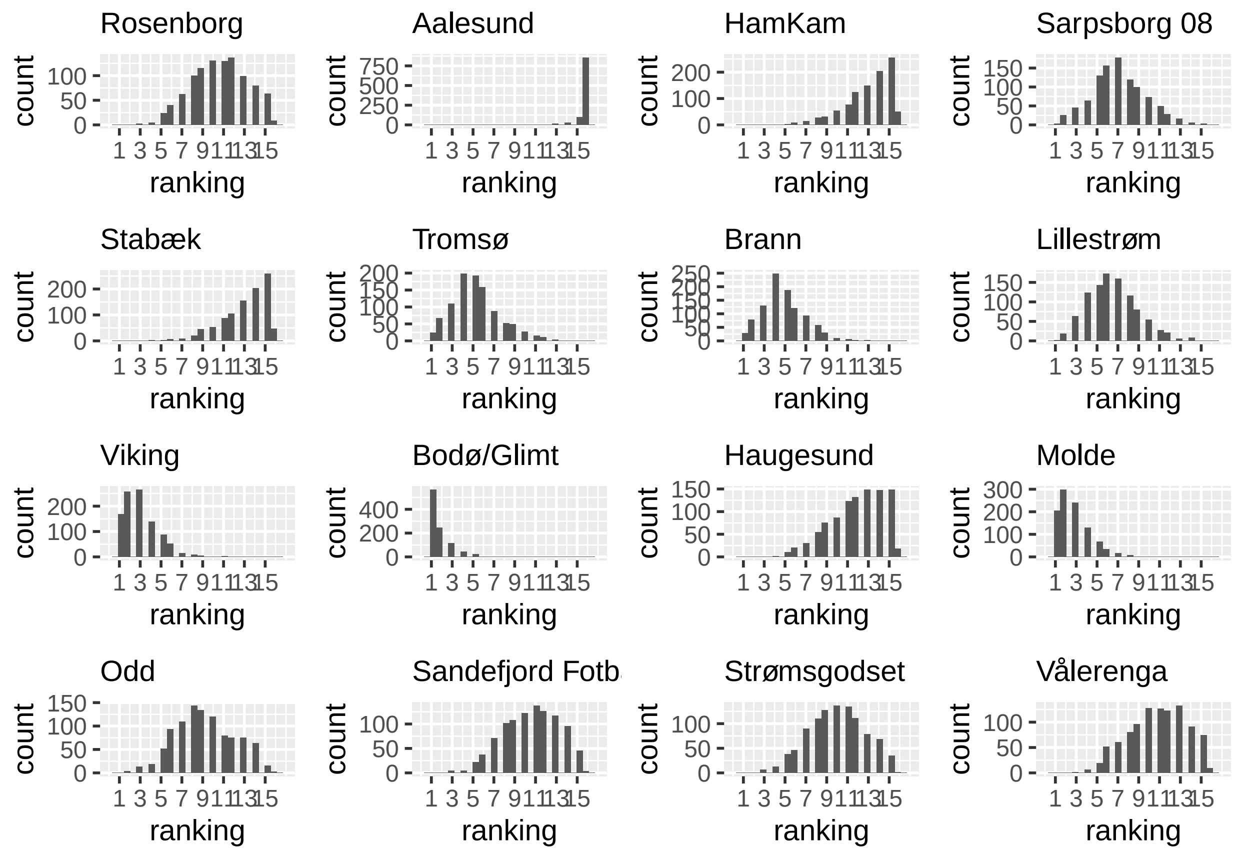
If we compare the ranking to the one we found in b), we see that the two tables are different - however most of the teams are ranked within 1-2 places away from their ranking in b).

d)

We now move on to a larger-scale simulation, where we simulate all matches for 1000 full seasons, using the strength parameters we found in c). For each season, we calculate the final rankings, and save these in a matrix.

```r
# d) Simulating 1000 seasons
n <- 1000
set.seed(420)
X <- setup_design(eliteserie)

# Create a new matrix which will be (240,5) with cols
eliteserie_sim <- eliteserie

# Calculate lambda_i for each row in design matrix
lambdas <- vector("numeric", length = length(X[, 1]))
for (j in 1:length(X[, 1])) {
    lambdas[j] <- exp(X[j, ] %*% poisson.fit)
}

# Matrix to contain placement for each team in every simulation,
# i.e. dim(n,16)
all_rankings <- matrix(nrow = n, ncol = 16)
colnames(all_rankings) <- unique(eliteserie$home)

for (i in 1:n) {

    # Simulate y_i from poisson dist. with lambda_i Convert design
    # matrix to dataframe similar to eliteserie Calculate
    # scoreboard from the new dataframe Save each team placement in
    # a dataframe with cols: team names and rows: simulation i

    y <- vector("numeric", length = length(X[, 1]))
    for (j in 1:length(X[, 1])) {
        y[j] <- rpois(1, lambdas[j])
    }
    eliteserie_sim$yh <- y[1:240]
    eliteserie_sim$ya <- y[241:480]
    rankings <- create_scoreboard(eliteserie_sim)

    for (k in 1:length(rankings[, 1])) {
        all_rankings[i, rankings$Team[k]] <- k
    }
}
```

For each team, we can now calculate their average placement over the 1000 seasons. A bar plot is shown for the distribution of placements for each team:



By applying the central limit theorem, we know that our sample means are approximately normally distributed - which we can see in the bar plots above. This allows us to construct confidence intervals by calculating the sample mean and sample standard deviation.

```
means_and_sds <- data.frame()

teams <- unique(eliteserie$home)
for (i in 1:length(teams)) {
    m <- mean(all_rankings[, i])
    std <- sd(all_rankings[, i])
    ci <- c(m - 1.96 * std/sqrt(length(all_rankings[, i])), m + 1.96 *
        std/sqrt(length(all_rankings[, i])))
    means_and_sds <- rbind(means_and_sds, c(teams[i], m, std, ci))
}
names(means_and_sds) <- c("Team", "Mean", "Standard error", "95% CI Lower",
    "95% CI Upper")
means_and_sds$`Standard error` <- as.numeric(means_and_sds$`Standard error`)
means_and_sds$`95% CI Lower` <- as.numeric(means_and_sds$`95% CI Lower`)
means_and_sds$`95% CI Upper` <- as.numeric(means_and_sds$`95% CI Upper`)
```

```r
means_and_sds <- means_and_sds[order(as.numeric(means_and_sds$Mean)),
    ]
row.names(means_and_sds) = NULL
print(means_and_sds, digits = 4)
```

```
##                  Team   Mean Standard error 95% CI Lower 95% CI Upper
## 1          Bodø/Glimt  1.722         1.0362        1.658        1.786
## 2               Molde  2.776         1.5356        2.681        2.871
## 3              Viking  3.023         1.6233        2.922        3.124
## 4               Brann  4.882         2.0631        4.754        5.010
## 5              Tromsø  5.307         2.3169        5.163        5.451
## 6          Lillestrøm  6.564         2.4132        6.414        6.714
## 7        Sarpsborg 08  7.106         2.5505        6.948        7.264
## 8                 Odd  9.186         2.8219        9.011        9.361
## 9        Strømsgodset  9.931         2.6984        9.764       10.098
## 10 Sandefjord Fotball 10.522         2.6576       10.357       10.687
## 11          Rosenborg 10.555         2.6788       10.389       10.721
## 12          Vålerenga 10.775         2.7184       10.607       10.943
## 13          Haugesund 11.856         2.5550       11.698       12.014
## 14             HamKam 13.001         2.2113       12.864       13.138
## 15             Stabæk 13.003         2.1872       12.867       13.139
## 16           Aalesund 15.791         0.5862       15.755       15.827
```

We see that Bodø/Glimt are the predicted winners of the season, with an average placing of 1.722. If we compare with the preliminary ranking found in b):

```r
kable(prelim_scoreboard, align = "l")
```

Team	Score	Goal difference
Viking	51	21
Bodø/Glimt	49	28
Tromsø	48	11
Brann	42	14
Molde	41	24
Lillestrøm	36	7
Sarpsborg 08	34	5
Odd	30	-3
Rosenborg	29	-6
Strømsgodset	27	-3
HamKam	24	-15
Vålerenga	21	-8
Sandefjord Fotball	21	-9
Haugesund	21	-14
Stabæk	17	-16
Aalesund	12	-36

We see that the top 5 teams are the same, however the placements are moved around. For accurately predicting this season's winner, one should consider running the simulations with only the remaining matches and not the entire season, as it might yield better predictions. However, to find the winner of future seasons, the approach of simulating the entire season should yield better results.