Smoking example continued

Re-download the `Smoking` data from the course webpage and fit a logistic regression model with covariates, age, age squared and smoking status (see practical sheet 6 for more details).

```
> file_path <- "http://www.statslab.cam.ac.uk/~rds37/teaching/statistical_modelling/"
> (Smoking <- read.csv(paste0(file_path, "Smoking.csv")))
> attach(Smoking)
> total <- Survived + Died
> propDied <- Died / total
> SmokingLogReg2 <- glm(propDied ~ Age.group + I(Age.group^2) + Smoker, family = binomial,
+ weights = total)
```

We now plot our estimate of the probability of dying as a function of age in black for non-smokers and in red for smokers. To do this, we use the `predict` function. When applied to a `glm` object, `predict` calls the `predict.glm` function. So type `?predict.glm` to get more details. In order to plot the expected proportion survived across a range of different ages, we form an artificial dataset of covariates and feed this as the `newdata` argument to `predict`. We form this dataset as follows

```
> newdata <- data.frame("Age.group" = rep(21:80, times=2),
+ "Smoker" = gl(2, 60, labels = levels(Smoker)))
> newdata[1:3, ]
```

View the help entries on `rep` and `gl` to understand what they do. It may help to experiment by supplying them with some arguments to see what they output.
To create the y values for our plot, we use `type = ''response''` in `predict`. This will give our estimates of the mean of the response for the different values of the covariates we have created.

```
> PredProp <- predict(SmokingLogReg2, newdata, type = "response")
```

The first half of the components of `PredProp` will correspond to smoking status equals `No`.

```
> plot(propDied[Smoker == "Yes"] ~ Age.group[Smoker == "Yes"],
+ xlab = "Age group", ylab = "Proportion died", col = "red")
> points(propDied[Smoker == "No"] ~ Age.group[Smoker == "No"], pch = 4)
> lines(21:80, PredProp[1:60], xlab = "Age", ylab = "Prob of dying", type = "l")
> lines(21:80, PredProp[61:120], col = "red")
```

To add pointwise confidence bands to our plot we do the following.

```
> PredLin <- predict(SmokingLogReg2, newdata, se.fit = TRUE, type = "link")
> str(PredLin)
```

This gives the fitted values of the linear predictor $x^T\hat{\beta}$ along with estimates for its standard deviation; these estimates will take the form

$$\sqrt{x^T i^{-1}(\hat{\beta})x}.$$

To transform the linear predictors to the scale of the mean response, we must apply the inverse of the link function. Here this will be

$$\eta \mapsto \frac{\exp(\eta)}{1+\exp(\eta)},$$

the inverse of the logit function. We code this as a function in R:

```
> invlogit <- function(x) exp(x) / (1 + exp(x))
```

Noting that asymptotically, the linear predictor is normally distributed, and that $\mathbb{P}(Z \leq 1.96) \approx 0.975$ when $Z \sim N(0,1)$, we plot our 95% pointwise confidence bands as follows:

```
> lines(21:80, invlogit(PredLin$fit[1:60] + 1.96 * PredLin$se.fit[1:60]), lty = 2)
> lines(21:80, invlogit(PredLin$fit[1:60] - 1.96 * PredLin$se.fit[1:60]), lty = 2)
> lines(21:80, invlogit(PredLin$fit[61:120] + 1.96 * PredLin$se.fit[61:120]),
+ lty = 2, col = "red")
> lines(21:80, invlogit(PredLin$fit[61:120] - 1.96 * PredLin$se.fit[61:120]),
+ lty = 2, col = "red")
```

Poisson regression

Download the English Premiership data from 2014 with

```
> detach(Smoking)
> football2014 <- read.csv(paste0(file_path, "football2014.csv"))
> football2014[1:3, ]
  GoalsScored         By        Against HomeAway
1           2    Arsenal Crystal Palace     Home
2           2  Leicester        Everton     Home
3           1 Man United        Swansea     Home
```

The first row says that Arsenal scored 2 goals against Crystal Palace when Arsenal was playing at home. There are 20 teams in the Premier League and each team plays every other team twice, once at home and once away. Thus 380 matches are played in total. Our dataset here has 524 rows as the goals scored by the home and away teams are recorded separately, and the season has not yet finished.

```
> football2014[263:265, ]
    GoalsScored             By    Against HomeAway
263           1 Crystal Palace    Arsenal     Away
264           2        Everton  Leicester     Away
265           2        Swansea Man United     Away
```

Row 263 gives data from the same match as that for the first row. The observation says that Crystal Palace scored 1 goal against Arsenal when Crystal Palace was playing away. If you wish, you can treat favourite team as the baseline by doing

```
> football2014$By <- relevel(football2014$By, "Man United")
> football2014$Against <- relevel(football2014$Against, "Man United")
```

for example. Here this forces Manchester United to be the first level in each of the factors so it is used as the reference level in the default corner point constraints used by R. Write down the model being fitted by the following command. Note that R uses the canonical log link by default

```
> attach(football2014)
> LogLinMod <- glm(GoalsScored ~ HomeAway + By + Against, family = poisson)
> summary(LogLinMod)
```

Try to interpret the results of the output. What is the size of the home advantage and is it statistically significant? (Note we should view any inferential statements with some reservation as there are several issues here including the fact that the scores will not be independent)

The function `barplot` can be useful to visualise the results of the fit e.g.

```
> attack_strength <- exp(sort(coef(LogLinMod)[3:21], decreasing = TRUE))
> barplot(rev(attack_strength), las=2, horiz=TRUE, cex.names=0.75)
```

The options `las=2` and `cex.names=0.75` rotate the labels so they are perpendicular to the axes and reduce their font size respectively.
The following analyses are optional, though I hope interesting. We will attempt to find for each team, the probability that at the end of the season they are in position $j = 1, \ldots, 20$, based on our fitted model. Download the remaining fixtures from the course webpage using

```
> detach(football2014)
> fixtures_remaining <- read.csv(paste0(file_path, "fixtures_remaining.csv"))
```

According to our model, the number of goals scored by each team in each match are independent Poisson random variables. Our estimates of the means of these Poisson random variables can be obtained using the `predict` function:

```
> Pred <- predict(LogLinMod, newdata=fixtures_remaining, type="response")
> cbind(fixtures_remaining, Pred)
```

Using these means, we can simulate the scores of the remaining matches in the premiership. In each match, a team is awarded 3 points if it wins, 1 if it draws and 0 if it loses. The final positions of the teams at the end of the season are based on the total number of points accrued. Issue the following code that creates a matrix of 1000 simulated versions of the points gained by each of the teams in the remaining matches (you may wish to copy and paste).

```
# number of simulations
B <- 1000
n_rem_fix <- length(Pred)/2

# create an empty matrix to store the simulation results
sim_points <- matrix(nrow=2*n_rem_fix, ncol=B)

for (b in 1:B) {
  # The simulated difference in the score between the Home and Away teams
  sim_score_diff <- rpois(n=n_rem_fix, lambda=Pred[1:n_rem_fix]) -
    rpois(n=n_rem_fix, lambda=Pred[(n_rem_fix+1):(2*n_rem_fix)])

  # Calculate the points scored
  points_scored <- 2*sign(sim_score_diff)
  points_scored <- c(pmax(points_scored+1, 0), pmax(1-points_scored, 0))
  sim_points[, b] <- points_scored
}
```

The `aggregate` function groups data by factor levels and then applies a given function to summarise each group. Here we use it to sum the points attained by each team in the matches it plays.

```
> sim_table <- aggregate(sim_points, by=list(fixtures_remaining$By), FUN=sum)
> sim_table[, 1:10]
> rownames(sim_table) <- sim_table[, 1]
> sim_table <- sim_table[, -1]
```

Now download the current standings of the teams with

```
> (table_cur <- read.csv(paste0(file_path, "table_cur.csv")))
```

and add these to the simulated points table

```
> sim_table <- sim_table + table_cur[, 2]
```

Recall that the `apply` function (starred section of Practical 4) applies a given function to each row or each column of a matrix so e.g. `apply(A, 1, mean)` computes the row means of a matrix `A`. The `rank` function returns the rank of each element of a vector. Here we use it where ties are broken at random.

```
> set.seed(1)
> rank(c(4, 6, 1, 2, 2, 7.9), ties.method="random")
[1] 4 5 1 3 2 6
```

Using `rank` on each column of `-sim_table` will now rank the teams in order of decreasing points.

```
> sim_ranks <- apply(-sim_table, 2, function(x) rank(x, ties.method="random"))
```

The `tabulate` function takes an integer-valued vector as its first argument, and counts the number of times each integer from 1 up to its second argument occurs in it.

```
> final_standings <- apply(sim_ranks, 1, function(x) tabulate(x, 20)) / B
```

`final_standings` now contains our estimated probability of each team being in each position at the end of the season. The matrix is perhaps easiest to view in its transposed form: `t(final_standings)`. A heatmap can be produced by

```
> heatmap(t(final_standings), Rowv = NA, Colv=NA)
```