hw2_code

Isaac Kleisle-Murphy

1/21/2022

```
knitr::opts_chunk$set(echo = TRUE)
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ISLR)
library(bestglm)
```

```
## Loading required package: leaps
```

```
library(e1071)
library(caret)
```

```
## Loading required package: lattice
```

```
### using a bunch of names, that I often forget
logit <- function(x){log(x/(1 - x))}
inverse_logit <- function(x){exp(x)/(1 + exp(x))}
sigmoid <- function(x){exp(x)/(1 + exp(x))}
```

4.6

a.)

As set forth in 4.3.1, the interpretation goes ~ "a one unit increase in weight induces a multiplicative impact of $\exp(0.5893) = 1.802726$ on the predicted mean, i.e. the mean at $x = weight + 1$ equals the mean at $weight$ multiplied by roughly 1.803.

To construct the "reported" (I'm taking this to mean reported in Table 4.8) intercept/coefficient specific Wald CI's, we follow the standard z-score procedure, taking `Estimate` $\pm z^*_{(.975)}\cdot$(`Std Error`), where $z^*_{(.975)} \approx 1.96$ satisfies $\Phi_{N(0,1)}(z^*_{(.975)}) = .975$. This is verified below:

```
z = qnorm(.975)
### intercept
print(c(-.4284 - z * .1789, -.4284 + z * .1789))
```

```
## [1] -0.77903756 -0.07776244
```

```
### weight
print(c(.5893 - z * .0650, .5893 + z * .0650))
```

```
## [1] 0.4619023 0.7166977
```

This matches the CI provided in the table.

b.)

For the Wald test, we can fix null $H_0 : \beta_w = 0$ against $H_1 : \beta_w \neq 0$ (two-sided), test against a χ^2_1, and compute

```
b0 = 0
z_stat = (.5893 - b0) / (.0650)
x_stat = z_stat^2
1 - pchisq(x_stat, 1)
```

```
## [1] 0
```

which returns a tiny p-value, and hence we reject. That is, we reject the hypothesis that X and Y are independent, as in fact the model shows evidence that there may be a linear relationship through the log odds (i.e. logistic regression appears suitable).

If we were to perform a LRT of y's independence of x, we might specify the "old" or "null" model to have the form (using LME shorthand) `y ~ 1`, while our new model `y ~ 1 + x`, and perform the Drop-In Deviance test of $Dev(New) - Dev(Old) \sim \chi^2_{[(N-1)-(N-2)]} = \chi^2_1$. However, in order to compute the old deviance, we would need the deviance of the old model, i.e. the deviance of a model that is just an intercept. To this end, the log-likelihood would suffice as well, since both deviance and log-likelihood are provided in the table (just need to take the difference between the appropriate pair).

c.)

This is a question of whether the data exhibits more variability/variance than the Poisson would allow (under it's mean = variance construction). First, we see that the deviance (~560) greatly outpaces the DF (171), which is a preliminary indicator of overdispersion. Moreover, looking at Table 4.4, we see that the bucket variances are generally larger than the bucket means – another sign of overdispersion, given that the Poisson should have mean equal to variance.

4.12

a.)

Setting up the drop-in-deviance test elucidates the degrees of freedom. Specifically, under the full/saturated model, we have $df_s = 7 \cdot 3 \cdot 3 = 63$, as there is a parameter for each data point in the table. However, for the main model, we have $df_m = 6 + 2 + 2 + 1 = 11$ total parameters (interval, histology, state, plus an intercept α). Hence, the deviance takes `df` equal to $df_s - df_m = 63 - 11 = 52$. In terms of model adequacy, we see heuristically that deviance is less than degrees of freedom – this is sure to give us a chi-squared test with a large p-value, and hence model is likely appropriate.

Since

b.)

We might interpret $\hat{\beta}_2^S - \hat{\beta}_1^S$ as the log of the multiplicative change in predicted mean resulting from moving from state 1 to state 2, provided everything else is held constant. We see this because assuming all other features are held constant, the ratio between the predicted mean in state 2 and predicted mean in state 1 is given by $\exp(\beta_2^S \delta(s = 2) + FIXED)/\exp(\beta^S \delta(s = \text{`}) + FIXED) = \exp(\hat{\beta}_2^S - \hat{\beta}_1^S)$. In other words, exponentiate this difference, and that is how much you should multiply your old prediction for $s = 1$ by to get to your new prediction for $s = 2$.

An identical interpretation holds for $s = 3, s = 1$.

c.)

We construct a drop in deviance test, i.e.

- $df_{new} = 48; DEV_{new} = 41.5$
- $df_{old} = 52; DEV_{old} = 43.9$

Hence, we test statistic $43.9 - 41.5 = 2.4$ against a χ_4^2,

```
1 - pchisq(43.9 - 41.5, 52 - 48)
```

```
## [1] 0.6626273
```

This returns a large value; we'd likely likely fail to reject any hypothesis test. There's little evidence to suggest the model is significantly improved under this new interacted form.

4.23

a.)

For $\pi(x; \alpha, \beta) = \frac{1}{2} + \frac{1}{\pi} atan(\alpha + \beta x)$, we have, using $atan'(z) = \frac{1}{1+z^2}$ and the chain rule:

$$\pi'(x; \alpha, \beta) = \frac{1}{\pi} \cdot \frac{\beta}{1 + (\alpha + \beta x)^2}.$$

Thus, we see that when $\alpha = 0, \beta = 1$, we have

$$\pi'(x; \alpha = 1, \beta = 1) = \frac{1}{\pi} \cdot \frac{1}{1 + x^2} \propto \frac{1}{1 + x^2},$$

which is a standard Cauchy kernel. More generally, the fundamental similarity between

$$\pi'(x;\alpha,\beta) = \frac{1}{\pi} \cdot \frac{\beta}{1+(\alpha+\beta x)^2}$$

and

$$p_{Cauchy}(x;\mu,\gamma) = \frac{1}{\pi} \cdot \frac{1}{\gamma\left(1+\left(\frac{x-\mu}{\gamma}\right)^2\right)} = \frac{1}{\pi} \cdot \frac{1}{\gamma+\frac{1}{\gamma}(x-\mu)^2}$$

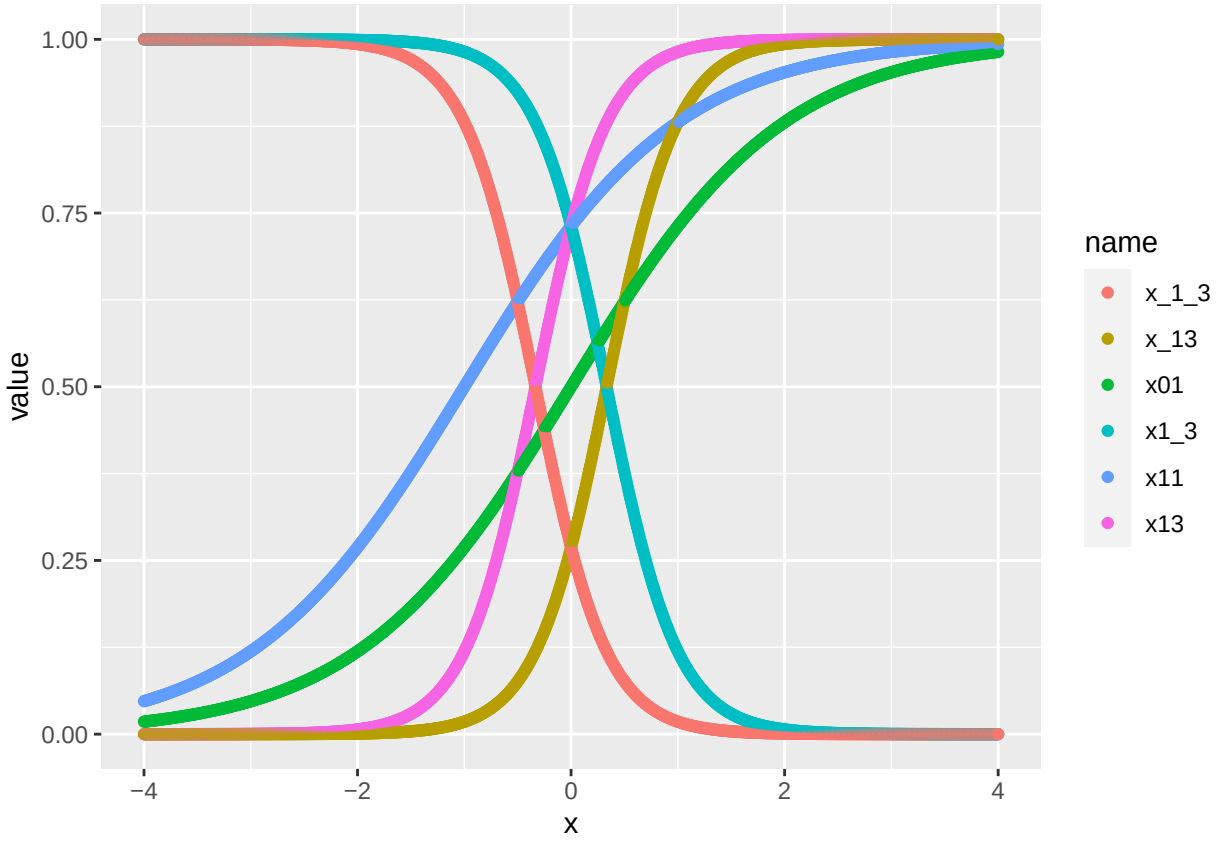
is evident by kernel-like pattern-matching.

To get a sense of the behavior of these respective transformations, let's first plot a couple toy examples, for comparison of standard logistic $\sigma(x;\alpha,\beta) = \frac{\exp(\alpha+\beta x)}{1+\exp(\alpha+\beta x)}$ and our existing $\pi(x;\alpha,\beta)$. Note that in the plots that follow, `x01` refers to $0+\beta x$; `x11` refers to $1+x$; `x13` refers to $1+3x$; `x1_3` refers to $1-3x$; `x_13` refers to $-1+3x$; and `x_1_3` refers to $-1-3x$. In short, the numbers correspond to α, β respectively, and a preceding underscore indicates negation.

```
###
# as given by problem
atan_423 <- function(x){1/2 + (1 / pi) * atan(x)}

### try out a couple different ones
xvec = seq(-4, 4, length.out=1000)

example_logistic = data.frame(
  x = xvec,
  x01 = inverse_logit(xvec),
  x11 = inverse_logit(1 + xvec),
  x13 = inverse_logit(1 + 3 * xvec),
  x1_3 = inverse_logit(1 - 3 * xvec),
  x_13 = inverse_logit(-1 + 3 * xvec),
  x_1_3 = inverse_logit(-1 - 3 * xvec)
) %>%
  tidyr::pivot_longer(., c("x01", "x11", "x13", "x1_3", "x_13", "x_1_3"))


example_atan = data.frame(
  x = xvec,
  x01 = atan_423(xvec),
  x11 = atan_423(1 + xvec),
  x13 = atan_423(1 + 3 * xvec),
  x1_3 = atan_423(1 - 3 * xvec),
  x_13 = atan_423(-1 + 3 * xvec),
  x_1_3 = atan_423(-1 - 3 * xvec)
) %>%
  tidyr::pivot_longer(., c("x01", "x11", "x13", "x1_3", "x_13", "x_1_3"))
```

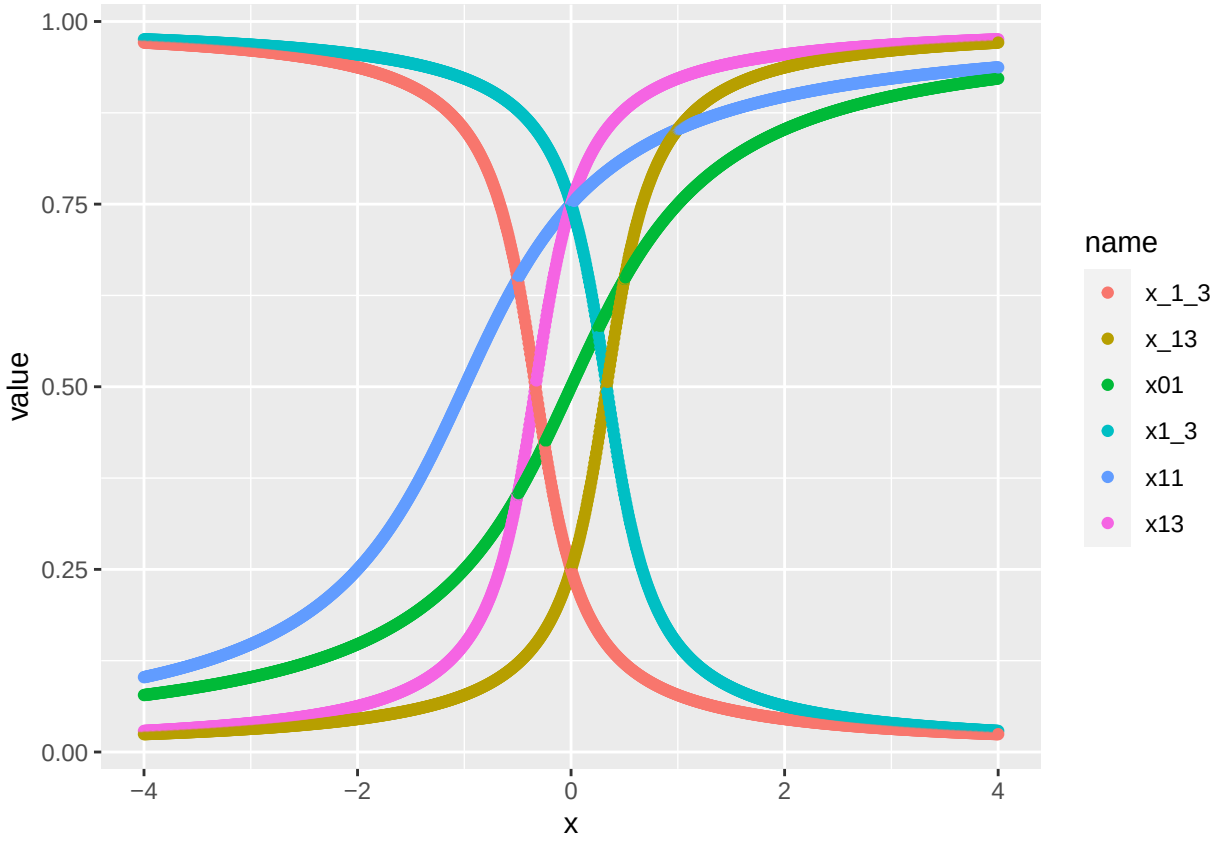
For the sigmoid CDF plot, we have:

```
ggplot(example_logistic, aes(x=x, y=value, color=name)) +
  geom_point()
```



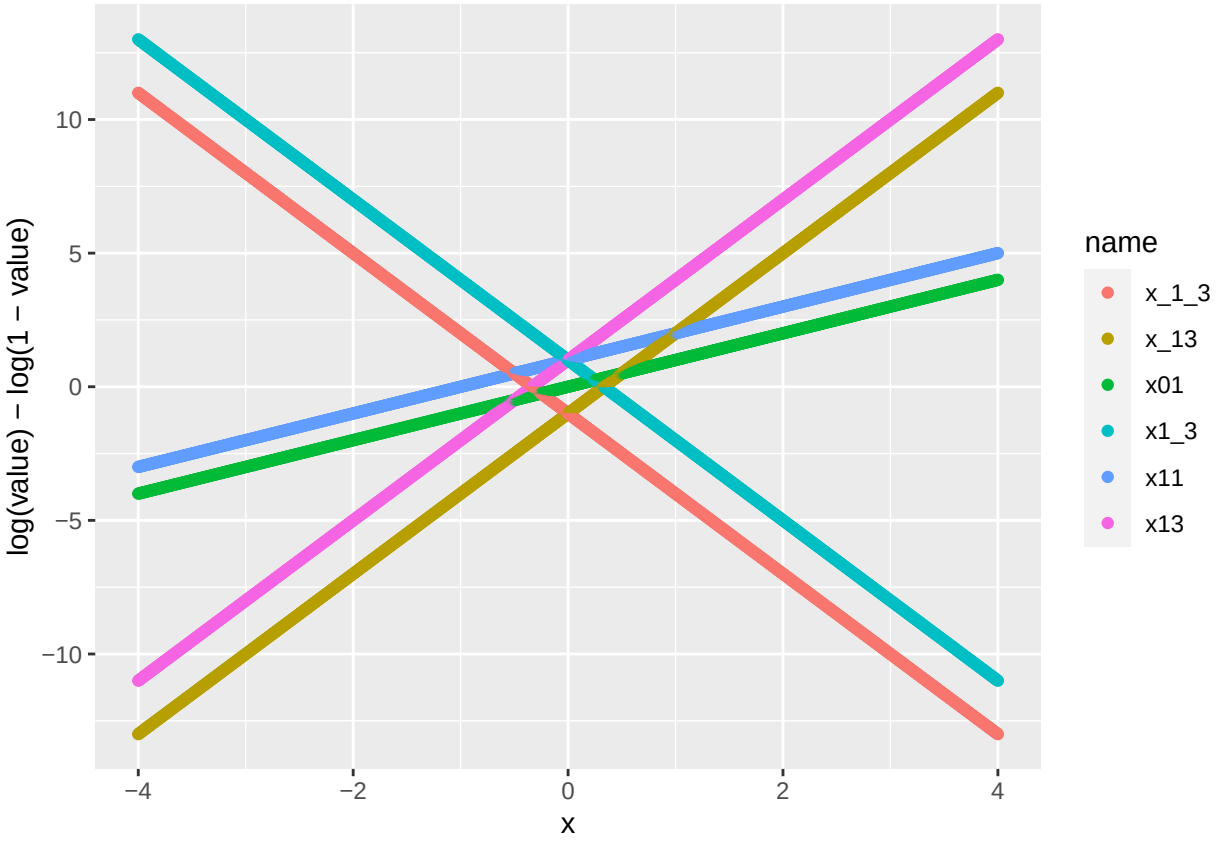
In general, these look loosely similar to the π-specified CDFS; though the sigmoid CDFs are a bit "sharper" in terms of their vertical climb and subsequent plateau.

```
ggplot(example_atan, aes(x=x, y=value, color=name)) +
  geom_point()
```



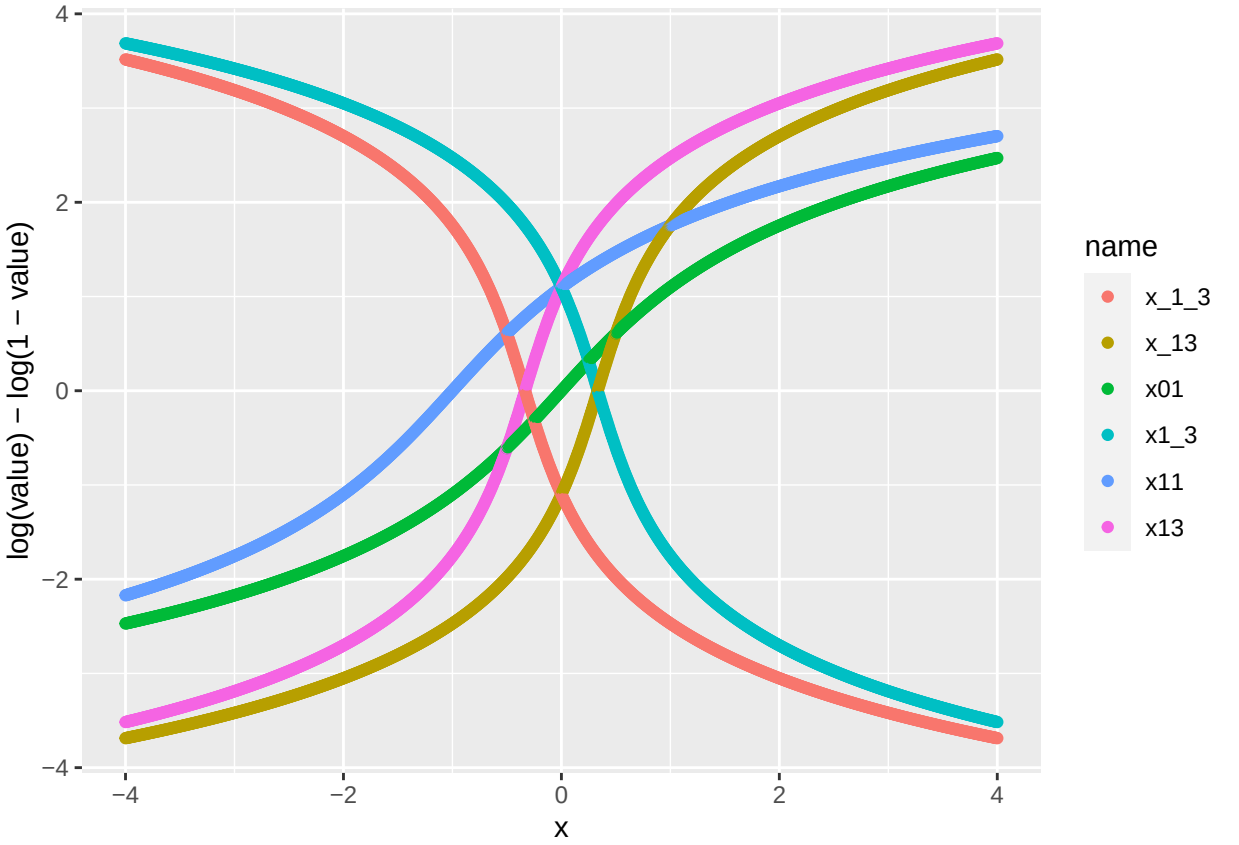
If we then look at $logit(\sigma(x;\alpha,\beta))$ and $logit(\pi'(x;\alpha,\beta))$ – that is, the log odds of each x under the respective models, things become clearer. Under the logistic, we see that log-odds are linear w.r.t. x, as we are keen to expect:

```
ggplot(example_logistic, aes(x=x, y=log(value) - log(1 - value), color=name)) +
  geom_point()
```



By contrast, log odds under π are not so linear – in fact, as x gets big, log odds begin to plateau and slow their rate of increase; similarly, as x gets small, log odds plateau and slow their rate of decrease. In other words, an "extreme" x produces a probability under the arctan/Cauchy that is less "extreme" than the probability under the logistic for that same "extreme" x. More succinctly, for extreme x, the probability is pushed to [0, 1] more slowly under Cauchy, due to the plateauing observed below:

```
ggplot(example_atan, aes(x=x, y=log(value) - log(1 - value), color=name)) +
  geom_point()
```



For cases in which the features x have fatter tails, the arctan/Cauchy construction might be preferable. If given a more outlying x (as may happen under fatter tails), the arctan/Cauchy function will not "overreact" and produce an extreme probability, which may be desirable.

b.)

We have shown that the provided CDF function reduces to a Cauchy kernel (shown going through the PDF); further, we know that the Cauchy is a Student T with `df=1`, which immediately shows that this is a special case of the Student T. Moreover, we also know that the Student-T tends towards the Gaussian as $df \to \infty$; we further know that a normal corresponds to the probit regression. In this way, we see a gamut of fat-tailedness for this general class of link functions – at the fattest-tailed end is the distribution specified in part a., while at the skinnier-tailed end is the normal that the Student T tends towards (as $df \to \infty$).

5.6

```
require(ggplot2)
### init data
shuttle_data = data.frame(
  ft=1:23,
  temp=c(
    66, 72, 70, 75, 75, 70, 73, 78, 70, 76, 69, 70,
    67, 81, 58, 68, 57, 53, 76, 67, 63, 67, 79
  ),
  td=c(
```

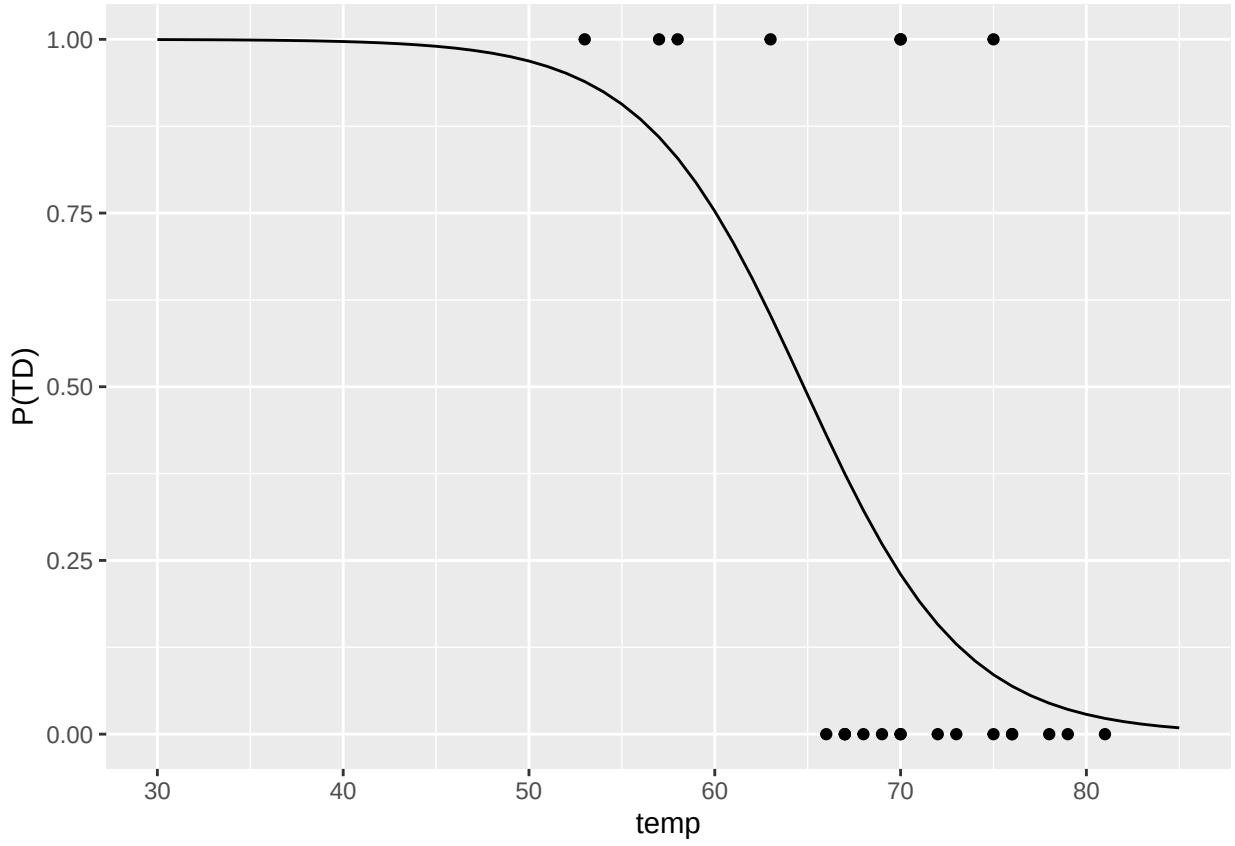
```
    0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0,
    0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0
  )
)

### fit
clf = glm(td ~ 1 + temp, data=shuttle_data, family=binomial())

### predict
preds = data.frame(temp=30:85)
preds$pi = predict(clf, newdata=preds, type="response")
preds$yhat = round(preds$pi)

### plot it
plt = ggplot(preds, aes(x=temp, y=pi)) +
  geom_path() +
  geom_point(data=shuttle_data, aes(x=temp, y=td)) +
  labs(y="P(TD)")

### coefs
theta = as.numeric(coef(clf))
theta
```

```
## [1] 15.0429016 -0.2321627
```

a.)

The plot is below. We see that as temperature decreases, the modeled probability of thermal distress increases. More specifically, for every one unit decrease in temperature, the log odds ratio for thermal distress increases by 0.2321627.

```
plt
```



b.)

Per the model, the probability of thermal distress at 31 degrees is roughly .9996. Of course, this is an extrapolation, but it speaks to the extreme condition experienced during flight.

```
preds %>% filter(temp == 31)
```

```
##   temp        pi yhat
## 1   31 0.9996088    1
```

c.)

Looking at the model, and aiming for a 95% confidence interval:

```
summary(clf)
```

```
##
## Call:
## glm(formula = td ~ 1 + temp, family = binomial(), data = shuttle_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.0611  -0.7613  -0.3783   0.4524   2.2175
```

```
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  15.0429     7.3786   2.039   0.0415 *
## temp         -0.2322     0.1082  -2.145   0.0320 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 28.267  on 22  degrees of freedom
## Residual deviance: 20.315  on 21  degrees of freedom
## AIC: 24.315
##
## Number of Fisher Scoring iterations: 5
```

the 95% CI for the coefficient is thus

```
c(
  theta[2] - qnorm(.975) * 0.1082,
  theta[2] + qnorm(.975) * 0.1082
)
```

```
## [1] -0.44423085 -0.02009464
```

Which in terms of direct odds is:

```
c(
  theta[2] - qnorm(.975) * 0.1082,
  theta[2] + qnorm(.975) * 0.1082
) %>% exp()
```

```
## [1] 0.6413174 0.9801059
```

Then, we do a Wald test on the coefficient, assuming $\alpha = .05$ and the usual zero/null hypothesis setup. Our p-value is

```
1 - pchisq(
  (theta[2] / 0.1082)^2,
  1
)
```

```
## [1] 0.03189839
```

At this α, the effect appears significant. In other words, we are inclined to conclude that temperature is not independent of thermal distress, i.e.there is some meaningful relationship between the two.

5.15

a.)

In order to maximize predicted probability, we seek to choose the x that maximizes $x^T\beta = -.8678x_{def} + 2.404x_{vic}$. This is achieved for $x = \langle 0, 1\rangle$, i.e. a black defendant and white victim, for a probability of

$$\frac{\exp(-3.5961 + 2.404)}{1 + \exp(-3.5961 + 2.404)} \approx .2329$$

b.) As this is a conditional odds ratio, we hold everything but for the variable of interest. Hence, given defendant race, we estimate with 95% confidence that the conditional odds ratio of the death penalty range between $\exp(1.3068 - 0) = 3.69$ and $\exp(3.7175 - 0) = 41.16$ for a white victim (`vic=1`). In the other direction, we estimate with 95% confidence that given victim race, the conditional odds ratio for a white defendant (`def=1`) range between $\exp(-1.5633 - 0) = .209$ and $\exp(-.1140 - 0) = .89$.

Using $\hat{\pi}_{ij}$ to mean the modeled probability when `def=i` and `vic=j`. Hence, the odds ratio is

$$\hat{\theta} = \frac{\hat{\pi}_{11}\hat{\pi}_{00}}{\hat{\pi}_{01}\hat{\pi}_{01}} = \frac{\frac{\exp(-3.5961+\beta_1+\beta_2)}{1+\exp(-3.5961+\beta_1+\beta_2)} \cdot \frac{\exp(-3.5961)}{1+\exp(-3.5961)}}{\frac{\exp(-3.5961+\beta_2)}{1+\exp(-3.5961+\beta_1)}\frac{\exp(-3.5961+\beta_2)}{1+\exp(-3.5961+\beta_1)}} = \frac{(\exp(-3.5961+\beta_1)+1)(\exp(-3.5961+\beta_2)+1)}{(\exp(-3.5961)+1)(exp(-3.5961+\beta_1+\beta_2))}.$$

c.)

The LR test is, using the table to get chi-squared sums, which rejects and indicates that with victim's race held constant, defendant's race has a non-zero coefficient.

```
1 - pchisq(5.01, 1)
```

```
## [1] 0.02520131
```

The Wald test is then

```
1 - pchisq(5.59, 1)
```

```
## [1] 0.0180633
```

rejecting and producing a similar conclusion as above. Either way, we're likely to conclude that defendant race contributes meaningfully to the model.

d.)

Using the Deviance entry in the "Criteria for Assessing Goodness of Fit" section of the SAS output, we see the Deviance = G^2 = .3798 and Pearson Chi-square = X^2 = .1978. Since both are less than degrees of freedom, we are inclined (by the rule of thumb) to conclude that fits reasonably well.

5.35

Before digging in here, let's re-derive the binomial likelihood equations. In the general setting where

$$y_i \sim Binom\left(n_i, \frac{\exp(\alpha + x_i^T\beta)}{1 + \exp(\alpha + x_i^T\beta)}\right) = Binom(n_i, \pi(\hat{\theta})_i).$$

As usual, the likelihood is,

$$\begin{aligned}\mathcal{L}(\hat{\theta}; y) &= \prod_{i=1}^{n} \left(\frac{\pi(\hat{\theta})_i}{1 - \pi(\hat{\theta})_i} \right)^{y_i} \cdot (1 - \pi(\hat{\theta})_i)^{n_i} \\ &= \prod_{i=1}^{n} \left(\frac{\frac{\exp(\alpha + x_i^T \beta)}{1+\exp(\alpha + x_i^T \beta)}}{\frac{1}{1+\exp(\alpha + x_i^T \beta)}} \right)^{y_i} \cdot \left(\frac{1}{1 + \exp(\alpha + x_i^T \beta)} \right)^{n_i} \\ &= \prod_{i=1}^{n} \exp\Big((\alpha + x_i^T \beta) y_i \Big) [1 + \exp(\alpha + x_i^T \beta)]^{-n_i}.\end{aligned}$$

Hence, log-likelihood is

$$\ell(\theta, y) = \sum_{i=1}^{n} (\alpha + x_i^T \beta) y_i - n_i \log(1 + exp(\alpha + x_i^T \beta))$$

and gradients are (using chain rule)

$$\nabla_\alpha = \sum_{i=1}^{n} y_i - n_i \frac{\exp(\alpha + x_i^T \beta)}{1 + \exp(\alpha + x_i^T \beta)} \cdot 1 = \sum_{i=1}^{n} y_i - n_i \pi(\hat{\theta})_i.$$

and

$$\nabla_\beta = \sum_{i=1}^{n} x_i y_i - n_i \Big(\frac{\exp(\alpha + x_i^T \beta)}{1 + \exp(\alpha + x_i^T \beta)} \Big) x_i = \sum_{i=1}^{n} x_i y_i - x_i n_i \pi(\hat{\theta})_i.$$

When we set these to zero, as one does with MLE, we have

$$\sum_{i=1}^{n} y_i = \sum_{i=1}^{n} n_i \pi(\hat{\theta})_i$$

and

$$\sum_{i=1}^{n} x_i y_i = \sum_{i=1}^{n} x_i n_i \pi(\hat{\theta})_i$$

These will be used frequently in 5.35 and 5.39

a.)

Begin with the binary kernel:

$$\begin{aligned}\ell(\hat{p}i_i; Y) &\propto \sum_{i=1}^{n} \sum_{j=1}^{n_j} y_{ij} \log \hat{\pi}_i + (1 - y_{ij}) \log(1 - \hat{\pi}_i) \\ &\propto \sum_{i=1}^{n} \Big(\log \hat{\pi}_i \sum_{j=1}^{n_j} y_{ij} + \log(1 - \hat{\pi}_i) \sum_{j=1}^{n_j} (1 - y_{ij}) \Big) \\ &\propto \sum_{i=1}^{n} \Big(\log \hat{\pi}_i y_i + \log(1 - \hat{\pi}_i)(n_i - y_i) \Big)\end{aligned}$$

, which is the familiar binomial kernel

b.)

Recall the saturated estimators, i.e.

$$\hat{y}_{i,j,sat} = y_i,$$

which guarantee that for any i, j

$$y_{i,j} \log \hat{y}_{i,j,sat} + (1 - y_{i,j}) \log(1 - \hat{y}_{i,j,sat}) = 0$$

as one term zeros out due to the y_i and the other term zeros out due to a $\log(1)$. Hence, the overall likelihood is zero, despite the fact that there are a total of n saturated parameters – one for each observation.

By contrast, the saturated estimators for the full binomial case are MLEs, i.e.

$$\hat{y}_i = \sum_{j=1}^{n_i} y_{ij}/n_i = y_i/n_i$$

Critically, this $\hat{y}_i \geq 0$ – that is, it is no longer deterministically zero as before. Then, the saturated likelihood takes the form for each group

$$\underbrace{y_i \log(\hat{y}_i)}_{\geq 0} + \underbrace{(1 - y_i) \log(1 - \hat{y}_i)}_{\geq 0} \geq 0.$$

Now summing over all i, we see that the saturated likelihood here may be above 0 – much different than for the binary case. So depending on which saturation method we use, the deviance may be different.

c.)

As set forth above, the saturated likelihood – whether by the binomial or binarized method – is a strict function of the $y_1, y_2, \ldots$, however you wish to define them. Hence, when taking the difference in deviances between two functions, we take (using $\theta := [\alpha; \beta]$) as shorthand in non-saturated cases; it's just the parameterization)

$$DEV(\theta_1|\vec{y}) - DEV(\theta_2|\vec{y}) = 2\ell(\hat{\theta}_s; \vec{y}) - 2\ell(\hat{\theta}_1; \vec{y}) - 2\ell(\hat{\theta}_s; \vec{y}) + 2\ell(\hat{\theta}_2; \vec{y}) = 2\ell(\hat{\theta}_2; \vec{y}) - 2\ell(\hat{\theta}_1; \vec{y}),$$

where the $2\ell(\hat{\theta}_s; \vec{y})$ term is the same for both (depending on your data-entry method) and thus cancels as an effective constant. Hence, deviance does not depend on this value, as stated in the problem. In other words, whatever method you pick for saturated deviance, it will cancel out during the subtraction step, so it does not change the difference.

By the algebra in part b.), the $n_i = 1$ case reduces to a number of Bernoulli trials, and saturated likelihood is zero here. Hence, we have (using σ as the logistic function)

$$\begin{aligned}
DEV &= 2\ell(\hat{\theta}_s; \vec{y}) - 2\ell(\hat{\theta}_1; \vec{y}) \\
&= 0 - 2\ell(\hat{\theta}_1; \vec{y}) \\
&= -2 \sum_{i=1}^{n} y_i \log \hat{\pi} + (1 - y_i) \log(1 - \hat{\pi}) \\
&= -2 \sum_{i=1}^{n} y_i logit(\hat{\pi}) \log(1 - \hat{\pi}) \\
&= -2 \sum_{i=1}^{n} y_i (\hat{\alpha} + x_i^T \hat{\beta}) \log(1 - \sigma(\hat{\alpha} + x_i^T \hat{\beta}))
\end{aligned}$$

d.)

Now, picking up with the deviance from above, we know that (using the fixed point findings from the start of the problem)

$$\begin{aligned}
DEV &= -2\sum_{i=1}^{n} y_i \log\left(\frac{\pi(\hat{\theta})_i}{1-\pi(\hat{\theta})_i}\right) + \log(1-\pi(\hat{\theta})_i) - 2\sum_{i=1}^{n} y_i(\hat{\alpha} + x_i^T\hat{\beta}) + \log(1-\pi(\hat{\theta})_i) \\
&= -2\sum_{i=1}^{n} y_i\hat{\alpha} - 2\sum_{i=1}^{n} y_i x_i^T\hat{\beta} - 2\sum_{i=1}^{n}\log(1-\pi(\hat{\theta})_i) \\
&= -2\hat{\alpha}\sum_{i=1}^{n} y_i - 2\hat{\beta}^T\sum_{i=1}^{n} x_i y_i - 2\sum_{i=1}^{n}\log(1-\pi(\hat{\theta})_i) \\
&= -2\hat{\alpha}\sum_{i=1}^{n} n_i\pi(\hat{\theta})_i - 2\hat{\beta}^T\sum_{i=1}^{n} x_i n_i\pi(\hat{\theta})_i - 2\sum_{i=1}^{n}\log(1-\pi(\hat{\theta})_i) \\
&= -2\hat{\alpha}\sum_{i=1}^{n} 1\cdot\pi(\hat{\theta})_i - 2\hat{\beta}^T\sum_{i=1}^{n} x_i\cdot 1\cdot\pi(\hat{\theta})_i - 2\sum_{i=1}^{n}\log(1-\pi(\hat{\theta})_i),
\end{aligned}$$

which has no immediate y_i dependence, noting that $n_i = 1$.

5.39

a.)

As is given above, the likelihood is,

$$\begin{aligned}
\mathcal{L}(\hat{\theta}; y) &= \prod_{i=1}^{I}\left(\frac{\pi(\hat{\theta})_i}{1-\pi(\hat{\theta})_i}\right)^{y_i}\cdot(1-\pi(\hat{\theta})_i)^{n_i} \\
&= \prod_{i=1}^{I}\left(\frac{\frac{\exp(\alpha+x_i^T\beta)}{1+\exp(\alpha+x_i^T\beta)}}{\frac{1}{1+\exp(\alpha+x_i^T\beta)}}\right)^{y_i}\cdot\left(\frac{1}{1+\exp(\alpha+x_i^T\beta)}\right)^{n_i} \\
&= \prod_{i=1}^{I}\exp\Big((\alpha+x_i^T\beta)y_i\Big)[1+\exp(\alpha+x_i^T\beta)]^{-n_i}.
\end{aligned}$$

Hence, log-likelihood is

$$\ell(\theta, y) = \sum_{i=1}^{I}(\alpha + x_i^T\beta)y_i - n_i\log(1 + exp(\alpha + x_i^T\beta))$$

b.)

Starting with the original likelihood we found, we have

$$\mathcal{L}(\alpha,\beta;X,Y) = \prod_{i=1}^{I}\exp\Big((\alpha+x_i^T\beta)y_i\Big)[1+\exp(\alpha+x_i^T\beta)]^{-n_i} = \prod_{i=1}^{I}\exp\Big(\alpha y_i+y_i x_i^T\beta\Big)[1+\exp(\alpha+x_i^T\beta)]^{-n_i} = \exp\left(\alpha\sum_{i=1}^{I}y_i\right)\cdot\exp$$

Choosing $T(x,y) = \sum x_i y_i$, $g_\beta(z) = \exp\beta^T T(x,y)$, we have all two of the key elements of NFFC. Importantly, the $\prod[1+\exp(\alpha+x_i^T\beta)]^{-n_i}$ here simply serves to ensure that everything integrates to one, so once

$\exp \beta^T T(x,y)$ is chosen, the $[1+\exp(\alpha + x_i^T\beta)]^{-n_i}$ defaults to a normalizing constant. Hence, we see the Neyman-Fisher factorization, in which $T(x,y) = \sum x_i y_i$ is our sufficient statistic. This draws a parallel to the Cochran-Armitage test, which uses:

$$b = \frac{\sum_i n_i \left(\frac{y_i}{n_i} - \frac{\sum y_i}{n}\right)(x_i - \bar{x})}{\sum n_i (x_i - \bar{x})^2} = \frac{\sum_i \left(y_i - \frac{n_i \sum y_i}{n}\right)(x_i - \bar{x})}{\sum n_i (x_i - \bar{x})^2}.$$

Notice that this test, by the interaction term in the numerator, also hinges on a $\sum_i x_i y_i$, as the x and n terms are effectively fixed. In this way, both the likelihood and the C-A test have a critical dependence on $T(x,y)$ – it is this term that moves the needle w.r.t. both.

c.)

As we found at the start of 5.35, we have

$$\sum_{i=1}^{I} y_i = \sum_{i=1}^{I} n_i \pi(\hat{\theta})_i$$

and

$$\sum_{i=1}^{I} x_i y_i = \sum_{i=1}^{I} x_i n_i \pi(\hat{\theta})_i$$

substituting $\pi(\hat{\theta})_i = \frac{\exp(\alpha + x_i^T\beta)}{1+\exp(\alpha + x_i^T\beta)}$ yields

$$S = \sum_{i=1}^{I} y_i = \sum_{i=1}^{I} n_i \cdot \frac{\exp(\alpha + x_i^T\beta)}{1+\exp(\alpha + x_i^T\beta)}$$

and

$$\sum_{i=1}^{I} x_i y_i = \sum_{i=1}^{I} x_i n_i \cdot \frac{\exp(\alpha + x_i^T\beta)}{1+\exp(\alpha + x_i^T\beta)}$$

as desired.

d.)

Algebraically, the $\sum \hat{\mu}_i$ result follows directly from above. The second result is given by

$$\sum_{i=1}^{I} x_i y_i = \sum_{i=1}^{I} x_i n_i \pi(\hat{\theta})_i = \sum_{i}^{I} x_i \hat{\mu}_i$$

$$\Longrightarrow$$

$$\frac{\sum_{i=1}^{I} x_i y_i}{S} = \frac{\sum_i^I x_i \hat{\mu}_i}{S} = \frac{\sum_i^I x_i \hat{\mu}_i}{\sum_{j=1}^{I} \hat{\mu}_i},$$

using the first result. Intuitively, the first result makes some amount of sense – ordinarily, when taking an MLE for binomials, we take the sample mean. However here, under the logistic construction, we're just requiring that all of the row-wise predictions sum up to the total number – essentially, enforcing a final "global sample mean". In other words, probabilities must sum such that if you were doing plain, single-parameter p MLE on the entire dataset, the expectation under both the logistic and the single-parameter would be the same. The second result, using this quasi-"sample mean" to match the form of the sufficient statistic in part b. – in effect, using the quasi-"sample mean" to do moment-matching.

6.8

First, we enter the data:

```
throat_data = matrix(
  c(
   45, 0, 0,
   15, 0, 0,
   40, 0, 1,
   83, 1, 1,
   90, 1, 1,
   25, 1, 1,
   35, 0, 1,
   65, 0, 1,
   95, 0, 1,
   35, 0, 1,
   75, 0, 1,
   45, 1, 1,
   50, 1, 0,
   75, 1, 1,
   30, 0, 0,
   25, 0, 1,
   20, 1, 0,
   60, 1, 1,
   70, 1, 1,
   30, 0, 1,
   60, 0, 1,
   61, 0, 0,
   65, 0, 1,
   15, 1, 0,
   20, 1, 0,
   45, 0, 1,
   15, 1, 0,
   25, 0, 1,
   15, 1, 0,
   30, 0, 1,
   40, 0, 1,
   15, 1, 0,
   135, 1, 1,
   20, 1, 0,
   40, 1, 0
  ),
  byrow=T,
  ncol = 3
) %>%
  data.frame() %>%
  `colnames<-`(c("d", "t", "y")) %>%
  mutate(patient=row_number())
```

As Chapter 6 focuses heavily on AIC, we will do a best subsets model selection procedure (as there are only 2 features, so not too bad), and choose the subset of features that achieves the best AIC. We will allow up to first order interaction in the consideration of subsets.

```
### fit the models over relevant subsets
model_subsets = list(
  null=glm(y ~ 1, data=throat_data, family=binomial()),
  d=glm(y ~ 1 + d, data=throat_data, family=binomial()),
  t=glm(y ~ 1 + t, data=throat_data, family=binomial()),
  d_t=glm(y ~ 1 + d + t, data=throat_data, family=binomial()),
  dt=glm(y ~ 1 + d * t, data=throat_data, family=binomial())
)


###
lapply(model_subsets, function(clf) clf$aic) %>% unlist()
```

```
##     null        d        t      d_t       dt
## 48.17981 37.65134 46.57757 36.13794 36.32105
```

We see that the model of the form `y ~ 1 + d + t` (i.e. all features but no interaction) achieved the lowest AIC; hence it's the one we'll go with. The model is summarized as follows:

```
summary(model_subsets$d_t)
```

```
##
## Call:
## glm(formula = y ~ 1 + d + t, family = binomial(), data = throat_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3802  -0.5358   0.3047   0.7308   1.7821
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.41734    1.09457  -1.295  0.19536
## d            0.06868    0.02641   2.600  0.00931 **
## t           -1.65895    0.92285  -1.798  0.07224 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 46.180  on 34  degrees of freedom
## Residual deviance: 30.138  on 32  degrees of freedom
## AIC: 36.138
##
## Number of Fisher Scoring iterations: 5
```

In terms of interpretation/inference, we have that

- Use of tracheal tube increased (duration held constant) the odds of soreness by a multiplier of $exp(-1.65) = .192$; this would suggest that the tracheal tube potentially reduced soreness. As evidenced by the conditional odds for this feature, the multiplier's 95% CI ranged from roughly $[exp(-1.65895 - 2 * .92285), exp(-1.65895 + 2 * .92285)] = [0.03005729, 1.20532592]$, providing weak evidence of soreness reduction, with all other features held constant. So while the estimate indicates reduction, the wide CI limits our confidence in this finding.

- A one minute increase in duration of the surgery increased (tracheal tube use held constant) by a multiplier of $\exp(.06868) = 1.071093$, suggesting (somewhat intuitively) that the longer (and possibly more complicated?) one's surgery went, the more soreness was to be expected. Morevoer, the 95% CI here ranged from roughly $[exp(.06868 - 2 * .02641), exp(.06868 + 2 * .02641)] = [1.015986, 1.129189]$. As 1 is now outside of this CI, this provides much stronger evidence that increasing surgery duration (all else constant) increases the odds of reported soreness.

6.26

Work here is formulated in Agresti 6.6.4-6.6.6, for reference.

First, as this is a treatment vs. no treatment study, it is natural to position M_0 (the null model) as one that presumes multinomial distributions that are independent of treatment. Given that the doctor hypothesizes probabilities of (call this the alternative M_1)

```
N = c(100, 100) # number in each experiment
### doc's predictions
# % 2 to make it a full joint density
M1 = matrix(
  c(.2, .2, .6,
    .3, .3, .4) / 2,
  byrow=T,
  ncol=3
)
M1
```

```
##      [,1] [,2] [,3]
## [1,] 0.10 0.10  0.3
## [2,] 0.15 0.15  0.2
```

it is natural to multiply the marginal probabilities to formulate the independence model, i.e.

```
### compute marginal independence model
# much simpler marginal calculation than this makes it look
marg_outcomes = M1 * N
marg_probs = colSums(marg_outcomes) / sum(marg_outcomes)
treat_probs = N / sum(N)
M0 = t(
  marg_probs %*% t(treat_probs)
)
M0
```

```
##       [,1]  [,2] [,3]
## [1,] 0.125 0.125 0.25
## [2,] 0.125 0.125 0.25
```

Now, as set forth in 6.6.5, we let $\pi(M_0)$ play the role of $\pi(M)$, and let $\pi(M_1)$ play the role of π. Then, following the formulae in 6.6.4, we have

$$\lambda_X = n \sum_i \frac{(\pi_i - \pi(M)_i)^2}{\pi(M)_i} :$$

```
lambda_X = sum((M1 - M0)^2 / M0)
lambda_X
```

```
## [1] 0.04
```

i.e. a noncentrality parameter w.r.t n of $\lambda_X(n) = .04_n$. Then, for G, it is similarly

$$\lambda_G = 2n \sum_i \pi_i \log\left(\frac{\pi_i}{\pi(M)_i}\right):$$

```
lambda_G = -2  * sum(
  M0 * log(M1 / M0)
)
lambda_G
```

```
## [1] 0.04082199
```

i.e. $\lambda_G(n) \approx 0.04082199n$.

Counting parameters, we expect $df = 4 - 2 = 2$. We get this because the alternative (the doc's model) has (3-1)*2 free parameters, whereas the null model has just 2; hence 4-2 = 2. With the non-centrality parameters λ_X, λ_G computed, we are free to proceed. Using the `ncp` argument for noncentrality passed to the `pchisq(), dchisq, ...` family in R, we can compute powers through inverse CDFs, according to $P(X^2_{v,k} > \chi^2_v(\alpha))$. The assumption is that large sample X^2 follows the noncentral chi-squared.

For X, we have power

```
DF = 2
critical_value = qchisq(.05, df=DF, lower.tail = F)
1 - pchisq(critical_value, df=DF, ncp=lambda_X * 200)
```

```
## [1] 0.7175643
```

while for G we have power.

```
1 - pchisq(critical_value, df=DF, ncp=lambda_G * 200)
```

```
## [1] 0.7269024
```

b.)

If we were inclined to test at .90 power, we would again solve with inverse CDFS. For X, we would have approximately

```
n_cand = seq(100, 500, length.out=10000)
### brute forcing it here
sapply(
  n_cand,
  function(n){
    1 - pchisq(critical_value, df=DF, ncp=lambda_X * n)
  }
) -> powers_x
n_cand[which(powers_x >= .9)[1]]
```

```
## [1] 316.3816
```

or 316 total samples – perhaps ~ 158 to a grouping. Likewise, for G, we would have approximately

```
sapply(
  n_cand,
  function(n){
    1 - pchisq(critical_value, df=DF, ncp=lambda_G * n)
  }
) -> powers_g
n_cand[which(powers_g >= .9)[1]]
```

```
## [1] 309.981
```

Or ~ 310 total samples; roughly 155 per grouping.

Modeling MPG

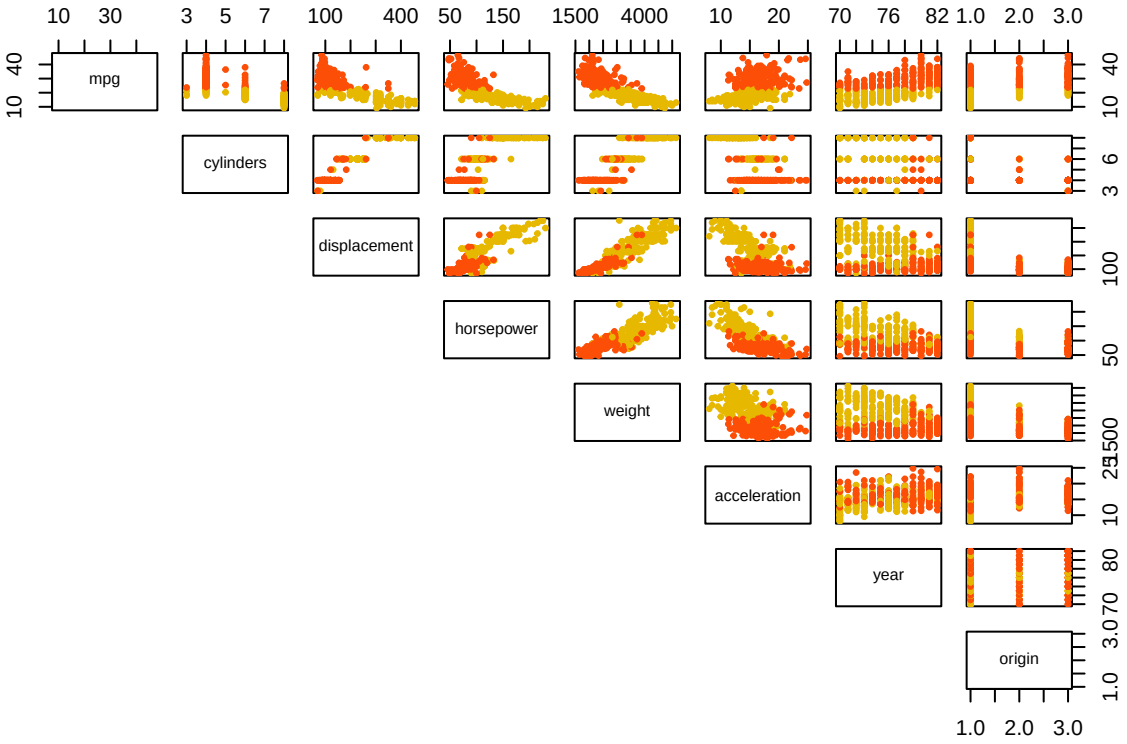
a.)

Engineering, per problem instructions:

```
auto = (ISLR::Auto) %>%
  mutate(y = as.integer(mpg > median(mpg)))
```

b.)

```
palette_hex = c("#E7B800", "#FC4E07")
pairs(
  auto %>% dplyr::select(mpg:origin),
  pch=19,
  cex=0.5,
  col=palette_hex[auto$y + 1],
  lower.panel=NULL
)
```



As the scatterplots here show (note orange corresponds to `Y=1`; yellow to `Y=0`), there appears to be a horizontal separating plane around 25 mpg. Similarly, displacement and weight also appear linearly separable w.r.t. `displacement`, `horsepower`, and `weight`. All of this linear separability suggests a simple logistic regression with linear coefficients may be an appropriate model choice here; hence it is pursued in the following sections.

c.)

```
### setup constants
FULL_FORMULA_STRING = "as.factor(y) ~ as.factor(cylinders) + displacement + horsepower + weight + accele

### split train/test
set.seed(1234)
### assign indices
train_idx = sample(1:nrow(auto), as.integer(392 / 2), replace=F)
test_idx = setdiff(1:nrow(auto), train_idx)
### do split
train_data = auto[train_idx, ]
test_data = auto[test_idx, ]
### fit baseline: note glm scales for us
clf_baseline = glm(as.formula(FULL_FORMULA_STRING), data=train_data, family="binomial", maxit=1e8)
### fit stepwise model
clf_step_forwards = step(clf_baseline, direction="forward")
```

```
## Start:  AIC=76.85
```

```
## as.factor(y) ~ as.factor(cylinders) + displacement + horsepower +
##     weight + acceleration + year + as.factor(origin)
```

```
clf_step_backwards = step(clf_baseline, direction="backward")
```

```
## Start:  AIC=76.85
## as.factor(y) ~ as.factor(cylinders) + displacement + horsepower +
##     weight + acceleration + year + as.factor(origin)
##
##                         Df Deviance     AIC
## - as.factor(origin)      2   55.329  75.329
## <none>                       52.855  76.855
## - displacement           1   56.195  78.195
## - acceleration           1   56.352  78.352
## - weight                 1   57.454  79.454
## - horsepower             1   61.991  83.991
## - as.factor(cylinders)   4   71.693  87.693
## - year                   1   80.263 102.263
##
## Step:  AIC=75.33
## as.factor(y) ~ as.factor(cylinders) + displacement + horsepower +
##     weight + acceleration + year
##
##                         Df Deviance    AIC
## - displacement           1   56.962 74.962
## <none>                       55.329 75.329
## - acceleration           1   58.910 76.910
## - weight                 1   59.289 77.289
## - horsepower             1   63.504 81.504
## - as.factor(cylinders)   4   74.435 86.435
## - year                   1   81.301 99.301
##
## Step:  AIC=74.96
## as.factor(y) ~ as.factor(cylinders) + horsepower + weight + acceleration +
##     year
##
##                         Df Deviance    AIC
## <none>                       56.962 74.962
## - weight                 1   59.364 75.364
## - acceleration           1   60.703 76.703
## - horsepower             1   65.269 81.269
## - as.factor(cylinders)   4   75.949 85.949
## - year                   1   82.256 98.256
```

```
clf_step_both = step(clf_baseline, direction="both")
```

```
## Start:  AIC=76.85
## as.factor(y) ~ as.factor(cylinders) + displacement + horsepower +
##     weight + acceleration + year + as.factor(origin)
##
##                         Df Deviance     AIC
## - as.factor(origin)      2   55.329  75.329
## <none>                       52.855  76.855
```

```
## - displacement          1   56.195  78.195
## - acceleration          1   56.352  78.352
## - weight                1   57.454  79.454
## - horsepower            1   61.991  83.991
## - as.factor(cylinders)  4   71.693  87.693
## - year                  1   80.263 102.263
##
## Step:  AIC=75.33
## as.factor(y) ~ as.factor(cylinders) + displacement + horsepower +
##     weight + acceleration + year
##
##                         Df Deviance    AIC
## - displacement           1   56.962 74.962
## <none>                       55.329 75.329
## + as.factor(origin)      2   52.855 76.855
## - acceleration           1   58.910 76.910
## - weight                 1   59.289 77.289
## - horsepower             1   63.504 81.504
## - as.factor(cylinders)   4   74.435 86.435
## - year                   1   81.301 99.301
##
## Step:  AIC=74.96
## as.factor(y) ~ as.factor(cylinders) + horsepower + weight + acceleration +
##     year
##
##                         Df Deviance    AIC
## <none>                       56.962 74.962
## + displacement           1   55.329 75.329
## - weight                 1   59.364 75.364
## - acceleration           1   60.703 76.703
## + as.factor(origin)      2   56.195 78.195
## - horsepower             1   65.269 81.269
## - as.factor(cylinders)   4   75.949 85.949
## - year                   1   82.256 98.256
```

```
### predict baseline
test_data$yhat_baseline = round(predict(clf_baseline, newdata=test_data, type="response"))
test_data$yhat_forwards = round(predict(clf_step_forwards, newdata=test_data, type="response"))
test_data$yhat_backwards = round(predict(clf_step_backwards, newdata=test_data, type="response"))
test_data$yhat_both = round(predict(clf_step_both, newdata=test_data, type="response"))

### look for disagreements
test_data %>%
  filter(
    (yhat_baseline + yhat_forwards + yhat_backwards + yhat_both) > 0,
    (yhat_baseline + yhat_forwards + yhat_backwards + yhat_both) < 4
  )
```

```
##      mpg cylinders displacement horsepower weight acceleration year origin
## 15  24.0         4          113         95   2372         15.0   70      3
## 166 20.0         8          262        110   3221         13.5   75      1
## 181 25.0         4          121        115   2671         13.5   75      2
## 208 20.0         4          130        102   3150         15.7   76      2
## 361 30.7         6          145         76   3160         19.6   81      2
```

```
## 362 25.4         6          168        116   2900         12.6   81      3
##                       name y yhat_baseline yhat_forwards yhat_backwards
## 15  toyota corona mark ii 1             1             1              0
## 166   chevrolet monza 2+2 0             0             0              1
## 181             saab 99le 1             0             0              1
## 208             volvo 245 0             0             0              1
## 361          volvo diesel 1             0             0              1
## 362       toyota cressida 1             1             1              0
##     yhat_both
## 15          0
## 166         1
## 181         1
## 208         1
## 361         1
## 362         0
```

Importantly, we see that on the test set, the models predict nearly identically: the baseline and forwards model always agree in their predictions, while the backwards and bidirectional models always agree in their predictions. As evidenced above, they disagree in six places. Hence, we report two sets of confusion matrices: one for the baseline and forwards pair, and one for the backwards and bidirectional pair, as the two sets predict identically on the test set.

d.) Confusion Matrices

For the baseline (full) and forwards models (as these two always agree), the confusion matrix is:

```
caret::confusionMatrix(
  reference=as.factor(test_data$y),
  data=as.factor(test_data$yhat_baseline)
)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 90 10
##          1 12 84
##
##                Accuracy : 0.8878
##                  95% CI : (0.835, 0.9283)
##     No Information Rate : 0.5204
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.7753
##
##  Mcnemar's Test P-Value : 0.8312
##
##             Sensitivity : 0.8824
##             Specificity : 0.8936
##          Pos Pred Value : 0.9000
##          Neg Pred Value : 0.8750
##              Prevalence : 0.5204
##          Detection Rate : 0.4592
```

```
##    Detection Prevalence : 0.5102
##       Balanced Accuracy : 0.8880
##
##        'Positive' Class : 0
##
```

We see that it returns 88.8% accuracy, a false positive rate of 0.1176471, and a true positive rate of r84/94‘.

Since the backwards + bidirectional models all predict 1/0 identically (see above), I'll report the confusion matrix once for both. For these models, the confusion matrix is:

```
caret::confusionMatrix(
  reference=as.factor(test_data$y),
  data=as.factor(test_data$yhat_backwards)
)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 88 10
##          1 14 84
##
##                Accuracy : 0.8776
##                  95% CI : (0.8233, 0.9199)
##     No Information Rate : 0.5204
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.7551
##
##  Mcnemar's Test P-Value : 0.5403
##
##             Sensitivity : 0.8627
##             Specificity : 0.8936
##          Pos Pred Value : 0.8980
##          Neg Pred Value : 0.8571
##              Prevalence : 0.5204
##          Detection Rate : 0.4490
##    Detection Prevalence : 0.5000
##       Balanced Accuracy : 0.8782
##
##        'Positive' Class : 0
##
```

We see that it returns 87.7% accuracy, a false positive rate of 0.1372549, and a true positive rate of r84/94‘ again. Hence, we might argue that the Stepwise model tests slightly worse.

More Matched Case Control

a.)

Our supposition here is that smoke is conditionally independent of C|A, S. In other words, we have

$$P(T|S, A) = P(T|S, A, C).$$

Now, recall our result from last class, that

$$\hat{\theta} \to_P \frac{P(T=0|C=0)\sum_s \sum_a P(S=s, A=a|C=0)P(T=1|S=s, A=a, C=0)}{\sum_s \sum_a P(S=s, A=a|C=0)P(T=0|S=s, A=a, C=1)P(T=1|C=0)}.$$

Leveraging our conditional independence assumption, combining conditionals into joint statements, and then marginalizing out, we then have:

$$\frac{P(T=0|C=0)\sum_s \sum_a P(S=s, A=a|C=0)P(T=1|S=s, A=a, C=0)}{\sum_s \sum_a P(S=s, A=a|C=0)P(T=0|S=s, A=a, C=1)P(T=1|C=0)}$$
$$=$$
$$\frac{P(T=0|C=0)\sum_s \sum_a P(S=s, A=a|C=0)P(T=1|S=s, A=a)}{\sum_s \sum_a P(S=s, A=a|C=0)P(T=0|S=s, A=a)P(T=1|C=0)}$$
$$=$$
$$\frac{P(T=0|C=0)\sum_s \sum_a P(T=1, S=s, A=a|C=0)}{\sum_s \sum_a P(T=0, S=s, A=a|C=0)P(T=1|C=0)}$$
$$=$$
$$\frac{P(T=0|C=0)P(T=1|C=0)}{P(T=0|C=0)P(T=1|C=0)}$$
$$=$$
$$1.$$

Hence,

$$\hat{\theta} \to_P 1.$$

b.)

The result follows directly from Bayes' rule, i.e. $P(A|B) = P(A, B)/P(B)$. First, we specify the conditional odds ratio, and simplify out

$$\begin{aligned}
&\frac{P(C=0, T=0|S=s, A=a)P(C=1, T=1|S=s, A=a)}{P(C=1, T=0|S=s, A=a)P(C=0, T=1|S=s, A=a)} \\
=&\frac{\frac{P(C=0,T=0,S=s,A=a)}{P(S=s,A=a)}\frac{P(C=1,T=1,S=s,A=a)}{P(S=s,A=a)}}{\frac{P(C=1,T=0,S=s,A=a)}{P(S=s,A=a)}\frac{P(C=0,T=1,S=s,A=a)}{P(S=s,A=a)}} \\
=&\frac{P(C=0, T=0, S=s, A=a)\cdot P(C=1, T=1, S=s, A=a)}{P(C=1, T=0, S=s, A=a)\cdot P(C=0, T=1, S=s, A=a)},
\end{aligned}$$

which has a clear dependence on S, A, as one would expect for an odds ratio conditional on it. In terms of preferences, we'd probably prefer the marginal if we're aiming to make higher-level/population-wide conclusions – if we're confident we've sampled fully from the population, the `sex` and `age` variables should be implicit/baked in, and we can make a more direct statement about `T` and `C`, without having to couch it in the conditional language of `S` and `A`. However, if we're interested in a prognosis for a particular person, the conditional version makes sense – it's a more personalized estimate for them.

c.)

Now, as we double-dip back into HW1, we reintroduce the homogenous association model, which for a set of indicator variables $V_1, \ldots, V_N$, takes the form

$$P(V_1, \ldots, V_N) \propto \exp(\sum_{j=1}^{N} \alpha_j V_j + \sum_{j \neq k} \alpha_{jk} V_j V_k).$$

Recall that the α_j are the indicator-specific coefficients, and the α_{jk} coefficients provide first-order interaction. Recall further from HW1 that for $V_1, \ldots, V_4$ we will have by constant-dropping

$$\begin{aligned} P(V_1, V_2 | V_3, V_4) &\propto P(V_1, V_2, V_3, V_4) \\ &= \exp(\sum_{j=1}^{N} \alpha_j V_j + \sum_{j \neq k} \alpha_{jk} V_j V_k) \\ &= \exp\left(\alpha_1 V_1 + \alpha_2 V_2 + \alpha_{12} V_1 V_2 + \sum_{j=3}^{4} (\alpha_{1j} V_1 V_j + \alpha_{2j} V_2 V_j)\right) \end{aligned}$$

Using `S`, `C`, `T`, and `A` now, we get:

$$\begin{aligned} &\frac{P(C=0, T=0 | S=s, A=a) \cdot P(C=1, T=1 | S=s, A=a)}{P(C=1, T=0 | S=s, A=a) \cdot P(C=0, T=1 | S=s, A=a)} \\ =&\frac{\exp(0) \cdot \exp\left(\alpha_C + \alpha_T + \alpha_{CT} + \alpha_{CS} s + \alpha_{CA} a + \alpha_{TS} s + \alpha_{TA} a\right)}{\exp\left(\alpha_C + \alpha_{CS} s + \alpha_{CA} a\right) \exp\left(\alpha_T + \alpha_{TS} s + \alpha_{TA} a\right)} \\ =&\frac{\cdot \exp\left(\alpha_C + \alpha_T + \alpha_{CT} + \alpha_{CS} s + \alpha_{CA} a + \alpha_{TS} s + \alpha_{TA} a\right)}{\exp\left(\alpha_C + \alpha_{CS} s + \alpha_{CA} a + \alpha_T + \alpha_{TS} s + \alpha_{TA} a\right)} \\ =&\exp(\alpha_{CT}), \end{aligned}$$

just as in HW1. As required, there is no dependence on `S` or `A`, so this is one suitable model. Note that I could have included an intercept in this model, but it would cancel out straightforwardly.

d.)

Let's take inspiration from the homogenous association model for this problem. In the problem above, we showed that the conditional odds ratio induced by modeling

$$P(C, T | S, A) \propto \exp\left(\alpha_C C + \alpha_T T + \alpha_{CT} CT + \alpha_{CS} CS + \alpha_{CA} CA + \alpha_{TS} TS + \alpha_{TA} TA\right)$$

gave us a conditional odds ratio equal to $\exp(\alpha_{CT})$. Then, we see that if we shoot for $P(C=1 | T, S, A)$, we will have

$$P(C=1 | T, S, A) \propto \exp\left(\alpha_C + \alpha_T T + \alpha_{CT} T + \alpha_{CS} S + \alpha_{CA} A + \alpha_{TS} TS + \alpha_{TA} TA\right)$$

This looks an awful lot like a logistic regression – we just don't have a normalizing constant. Recalling that a $1 + \exp(\eta)$ appropriately standardizes the logistic regression, we can then get a valid probability with

$$P(C=1 | T, S, A) \propto \frac{\exp\left(\alpha_C + \alpha_T T + \alpha_{CT} TC + \alpha_{CS} CS + \alpha_{CA} A + \alpha_{TS} TS + \alpha_{TA} TA\right)}{1 + \exp\left(\alpha_C + \alpha_T T + \alpha_{CT} TC + \alpha_{CS} CS + \alpha_{CA} A + \alpha_{TS} TS + \alpha_{TA} TA\right)} = \frac{\exp\left(\alpha_C + \alpha_T T + \alpha_{CT} T +}{1 + \exp\left(\alpha_C + \alpha_T T + \alpha_{CT} T\right.}$$

Now, α_C is the intercept, and we've setup a logistic regression of the form `C ~ 1 + T + S + A + T*S + T*A`s proven in the previous problem, will return us our desired conditional odds ratio. In short, our work in 3 ended up handing us a logit model.

e.)

We get the desired result via the consistency of the MLE: we're fitting a logistic regression by MLE, and MLE's are consistent estimators, so eventually or logistic regression coefficients will converge to the true coefficients. This must then be true for the α_{TC} coefficient, so we can reasonably estimate via the logistic fit.

f.)

The homogenous association model makes some big assumptions – namely, that the logistic as formulated above is appropriate, that the interactions are exactly as specified above, that it's a sigmoid, and not a probit or Cauchy quantile/link function, and so forth. In fact, the true conditional independence model (as it is conditional) may depend on S, A; however, under the homogenous association model, those S,A terms simply fall off/margin out cleanly, as seen in the computations in e.). In removing this dependence on S, A for the purposes of computing conditional odds, we're taking a bigger leap of faith and making more assumptions about model structure/form, for the computational/sampling conveniences it provides.

4.23 Part II

a.)

Recall that in the Bernoulli case (saturated deviance is perfect) for some link g

$$\begin{aligned}
DEV(\beta|Y) &= -2\ell(\beta|X,Y) \\
&= -2\sum_{i=1}^{n}[Y_i \log \pi_i + (1-Y_i)\log(1-\pi_i)] \\
&= -2\sum_{i=1}^{n}\left[Y_i \cdot logit(\pi_i(\beta)) + \log(1-\pi_i(\beta))\right] \\
&= -2\sum_{i=1}^{n}\left[Y_i \cdot logit(g(x_i^T\beta)) + \log(1-g(x_i^T\beta))\right]
\end{aligned}$$

Then, using $logit'(x) = \frac{1}{x(1-x)}$, we have

$$\begin{aligned}
\nabla_\beta - \ell(\beta|X,Y) &= -\nabla_\beta \sum_{i=1}^{n}\left[Y_i \cdot logit(g(x_i^T\beta)) + \log(1-g(x_i^T\beta))\right] \\
&= -\sum_{i=1}^{n}\left[\frac{Y_i}{g(x_i^T\beta)(1-g(x_i^T\beta))}\cdot g'(x_i^T\beta)x_i - \frac{1}{1-g(x_i^T\beta)}\cdot g'(x_i^T\beta)x_i\right] \\
&= -\sum_{i=1}^{n}\left[\frac{Y_i}{g(x_i^T\beta)(1-g(x_i^T\beta))} - \frac{1}{1-g(x_i^T\beta)}\right]g'(x_i^T\beta)x_i \\
&= -\sum_{i=1}^{n}\left[\frac{Y_i}{g(x_i^T\beta)(1-g(x_i^T\beta))} - \frac{g(x_i^T\beta)}{g(x_i^T\beta)(1-g(x_i^T\beta))}\right]g'(x_i^T\beta)x_i \\
&= -\sum_{i=1}^{n}(Y_i - g(x_i^T\beta))\cdot\frac{g'(x_i^T\beta)}{g(x_i^T\beta)(1-g(x_i^T\beta))}x_i.
\end{aligned}$$

Note – just for fun, that when $g = \sigma$, we have $\sigma' = \sigma \cdot (1 - \sigma)$ and the $\frac{g'(x_i^T\beta)}{g(x_i^T\beta)(1-g(x_i^T\beta))} = 1$.

Next, for expectation of the Hessian, recall from lecture that

$$\begin{aligned} E_\beta[\nabla^2 DEV(\beta|Y)] &= 2VAR\beta[\nabla DEV(\beta|Y)] \\ &= 2\sum_{i=1}^{n} x_i x_i^T \frac{g'(x_i^T\beta)^2}{g(x_i^T\beta)(1-g(x_i^T\beta))} \\ &= 2\sum_{i=1}^{n} x_i x_i^T \frac{g'(x_i^T\beta)^2}{g(x_i^T\beta)(1-g(x_i^T\beta))} \\ &= 2X^T W\beta X, \end{aligned}$$

for $W_\beta = diag\left(\frac{g'(x_i^T\beta)^2}{g(x_i^T\beta)(1-g(x_i^T\beta))}\right)$. Admittedly, I've written this up poorly – it's best to think of $g := F$, the CDF, and $g' := f$, the density. Mostly just a note to self there.

1-3

```
clip <- function(x, eps=1e-7){
  x_ = x
  x_[x_ == 0.] = eps
  x_[x_ == 1.] = 1 - eps
  x_
}

compute_deviance <- function(X, Y, link, beta, eps=1e-7){
  #' Computes binary deviance for arbitrary link. See GLM slides 10.
  #'
  eta = as.numeric(
    X %*% beta
  )
  dev = -2 * sum(
    Y * log(
      link(eta) / clip(1 - link(eta))
    ) +
      log(1 - link(eta))
  )
  dev
}

compute_grad_deviance <- function(X, Y, link, deriv_link, beta, eps=1e-7){
  #' Computes gradient of deviance
  #'

  eta = as.numeric(
    X %*% beta
  )
  W_b = diag(
    deriv_link(eta)^2 /
      (
        clip(link(eta) * (1 - link(eta)))
      )
```

```
  )
  grad_dev = -2 * t(X) %*% W_b %*% (
    (Y - link(eta)) / deriv_link(eta)
  )
  as.numeric(grad_dev)
}

compute_hessian_deviance <- function(X, Y, link, deriv_link, beta){
  #' Computes gradient of deviance
  #'

  eta = as.numeric(
    X %*% beta
  )
  W_b = diag(
    deriv_link(eta)^2 /
      (
        clip(link(eta) * (1 - link(eta)))
      )
  )

  result = 2 * t(X) %*% W_b %*% X
  result
}


solve_fisher_scoring <- function(
  X,
  Y,
  link,
  # inverse_link,
  deriv_link,
  max_iter=50000,
  tol=1e-5
){

  # init beta and beta history log
  beta = rep(0, dim(X)[2])
  beta_log = list(beta)
  ### init bool + counter
  is_converged = F
  n_iter = 0

  for (i in 2:(max_iter + 1)){
    # print(i)
    ### perform update
    beta = beta -
      solve(
        compute_hessian_deviance(X, Y, link, deriv_link, beta)
      ) %*% compute_grad_deviance(
        X, Y, link, deriv_link, beta
      ) %>% as.numeric()
    ### save
```

```
    beta_log[[i]] = beta
    ### elementwise deltas
    delta = beta - beta_log[[i - 1]]
    ### check convergence
    if(sqrt(as.numeric(t(delta) %*% delta)) < tol){
      is_converged = T
      n_iter = i
      break
    }
    # return(beta)
  }

  cat("###### Fisher Scoring Complete ######\n")
  cat("Iterations: ", n_iter, "\n")
  cat("Converged: ", is_converged, "\n")
  cat("L2 Norm: ", sqrt(as.numeric(t(delta) %*% delta)), "\n")
  cat("#######################################\n\n")
  beta
}


solve_gradient_descent <- function(
  X,
  Y,
  link,
  # inverse_link,
  deriv_link,
  alpha=function(i){1e-3},
  max_iter=50000,
  tol=1e-5,
  verbose=F
){

  # init beta and beta history log
  beta = rep(0, dim(X)[2])
  beta_log = list(beta)
  ### init bool + counter
  is_converged = F
  n_iter = 0

  for (i in 2:(max_iter + 1)){
    # if (i %% 500 == 0){
    #   print(i)
    # }
    ### perform update
    delta_beta = alpha(i) * compute_grad_deviance(
        X, Y, link, deriv_link, beta
    ) %>% as.numeric()
    if (verbose & i %% 100 == 0){
      # print first five coefficients at iteration
      cat(i, ": ", paste0(beta[1:5], collapse=", "), "\n")
    }
    beta = beta - delta_beta
```

```
    ### save
    beta_log[[i]] = beta
    ### elementwise deltas
    delta = beta - beta_log[[i - 1]]
    ### check convergence
    if(sqrt(as.numeric(t(delta) %*% delta)) < tol){
      is_converged = T
      n_iter = i
      break
    }
    # return(beta)
  }

  cat("###### Gradient Descent Complete ######\n")
  cat("Iterations: ", n_iter, "\n")
  cat("Converged: ", is_converged, "\n")
  cat("L2 Norm: ", sqrt(as.numeric(t(delta) %*% delta)), "\n")
  cat("########################################\n\n")
  # return list fo purposes of writeup
  beta_log

}
```

1 - Fit

We first fit the logistic by Fisher scoring. Outputs are the regression's coefficients.

```
### spam data

spam_data = read.csv("/Users/IKleisle/Stanford/STATS305B/spam.csv")
X = spam_data %>%
  dplyr::select(A.1:A.57) %>%
  as.matrix() %>%
  unname()
### mean and center scale
X = scale(X)
### fix outliers by truncation at 5 SD
X[X > 5] = 5; X[X < -5] = -5
X = cbind(1, X)
Y = as.integer(spam_data$spam == "spam")

summary((spam_data$A.56 - mean(spam_data$A.56)) / sd(spam_data$A.56))
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.26257 -0.23692 -0.19074  0.00000 -0.04707 50.98651
```

As a routine preprocessing step, I mean/center scaled the features X. However, as evidenced above, certain points had extreme outliers – for instance, in the 56th column, 75% of the data was between -.23 and -.04, but the max was a whopping 50.98 (that's after scaling!). These outliers jeopardized the fit model, since they would push certain $x_i^T \beta$ to value that was large in magnitude, which under sigmoid transformation R would round to 0 or 1, resulting in divide by zeros in the W matrix and a broken Fisher iteration. As a remedy for this, I assigned all features with a z-score > 5 to be truncated at $+5$, and all features with a z-score < -5 to be truncated at -5. I think truncating at five standard deviations is reasonable here.

Note that the clipping procedure (also used to manage numeric issues) follows from the standard SKLearn practice: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.log_loss.html

```
#
#
glm(
  Y ~ .,
  data=data.frame(cbind(X[, 2:ncol(X)], Y)), # use the scaled one
  family=binomial()
) -> base_fit
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
solve_fisher_scoring(
  X,
  Y,
  link=sigmoid,
  deriv_link=function(x){sigmoid(x) * (1 - sigmoid(x))},
  max_iter=50000,
  tol=1e-5
)
```

```
## ###### Fisher Scoring Complete ######
## Iterations:  15
## Converged:  TRUE
## L2 Norm:  1.087657e-09
## ####################################
```

```
##  [1] -12.20999546  -0.12853354  -0.20765822   0.04511291   2.71314918
##  [6]   0.56506632   0.40045955   0.92383457   0.36505992   0.33856486
## [11]   0.19413333  -0.02017585  -0.14169695  -0.04154313   0.24568359
## [16]   0.37307408   0.80299512   0.44011654   0.08552715   0.08506595
## [21]   0.67492444   0.29968573   0.11447811   0.77434542   0.62526644
## [26]  -3.05368893  -0.93805265 -38.92800676   0.38333444  -1.25798300
## [31]  -0.17661959   0.02380024   0.80262098  -0.37981731  -0.15121610
## [36]  -1.19420842   0.39417592  -0.07387843  -0.12662798  -0.34379754
## [41]  -0.05738998 -18.77334178  -2.08664522  -0.34582209  -1.12256251
## [46]  -0.64364658  -1.16737503  -0.21591321  -1.08769512  -0.37168255
## [51]  -0.06164635  -0.02995782   0.99515628   1.25441309   0.72926046
## [56]   2.01789166   1.03913201   0.67278591
```

Above are the logistic coefficients under fisher scoring; fortunately, they match those of base fit. We can have confidence that our implementation is working adequately.

```
coef(base_fit)
```

```
##  (Intercept)           V1           V2           V3           V4           V5
## -12.20999546  -0.12853354  -0.20765822   0.04511291   2.71314918   0.56506632
##           V6           V7           V8           V9          V10          V11
##   0.40045955   0.92383457   0.36505992   0.33856486   0.19413333  -0.02017585
##          V12          V13          V14          V15          V16          V17
##  -0.14169695  -0.04154313   0.24568359   0.37307408   0.80299512   0.44011654
```

```
##          V18          V19          V20          V21          V22          V23
##   0.08552715   0.08506595   0.67492444   0.29968573   0.11447811   0.77434542
##          V24          V25          V26          V27          V28          V29
##   0.62526644  -3.05368893  -0.93805265 -38.92800676   0.38333444  -1.25798300
##          V30          V31          V32          V33          V34          V35
##  -0.17661959   0.02380024   0.80262098  -0.37981731  -0.15121610  -1.19420842
##          V36          V37          V38          V39          V40          V41
##   0.39417592  -0.07387843  -0.12662798  -0.34379754  -0.05738998 -18.77334179
##          V42          V43          V44          V45          V46          V47
##  -2.08664522  -0.34582209  -1.12256251  -0.64364658  -1.16737503  -0.21591321
##          V48          V49          V50          V51          V52          V53
##  -1.08769512  -0.37168255  -0.06164635  -0.02995782   0.99515628   1.25441309
##          V54          V55          V56          V57
##   0.72926046   2.01789166   1.03913201   0.67278591
```

Next, we do the same, albeit with a T-link function for `df=50`

```
solve_fisher_scoring(
  X,
  Y,
  link=function(x){pt(x, df=50)},
  deriv_link=function(x){dt(x, df=50)},
  max_iter=50000,
  tol=1e-5
)
```

```
## ###### Fisher Scoring Complete ######
## Iterations:  20
## Converged:  TRUE
## L2 Norm:  8.140135e-06
## ####################################

##  [1]  -5.228764394  -0.070678536  -0.131953202   0.033998161   1.558259300
##  [6]   0.327584807   0.227087655   0.440098417   0.179453400   0.187787918
## [11]   0.112966893   0.001606835  -0.087360723  -0.026114085   0.145172063
## [16]   0.253235136   0.457295761   0.221714512   0.055820510   0.040460190
## [21]   0.285335641   0.176844237   0.101951303   0.440284542   0.372023685
## [26]  -1.341137548  -0.636412118 -14.698111340   0.209367092  -0.717147493
## [31]  -0.118156851   0.002451834   0.494359681  -0.217177242  -0.105986412
## [36]  -0.659185354   0.206742008  -0.055974544  -0.045135553  -0.197696165
## [41]  -0.017698734 -10.496888941  -1.194527852  -0.189557344  -0.590842577
## [46]  -0.337520317  -0.581618108  -0.112472263  -0.595861593  -0.251654801
## [51]  -0.043658759  -0.023151551   0.536771176   0.630594868   0.463061161
## [56]   0.733593475   0.492442027   0.401497140
```

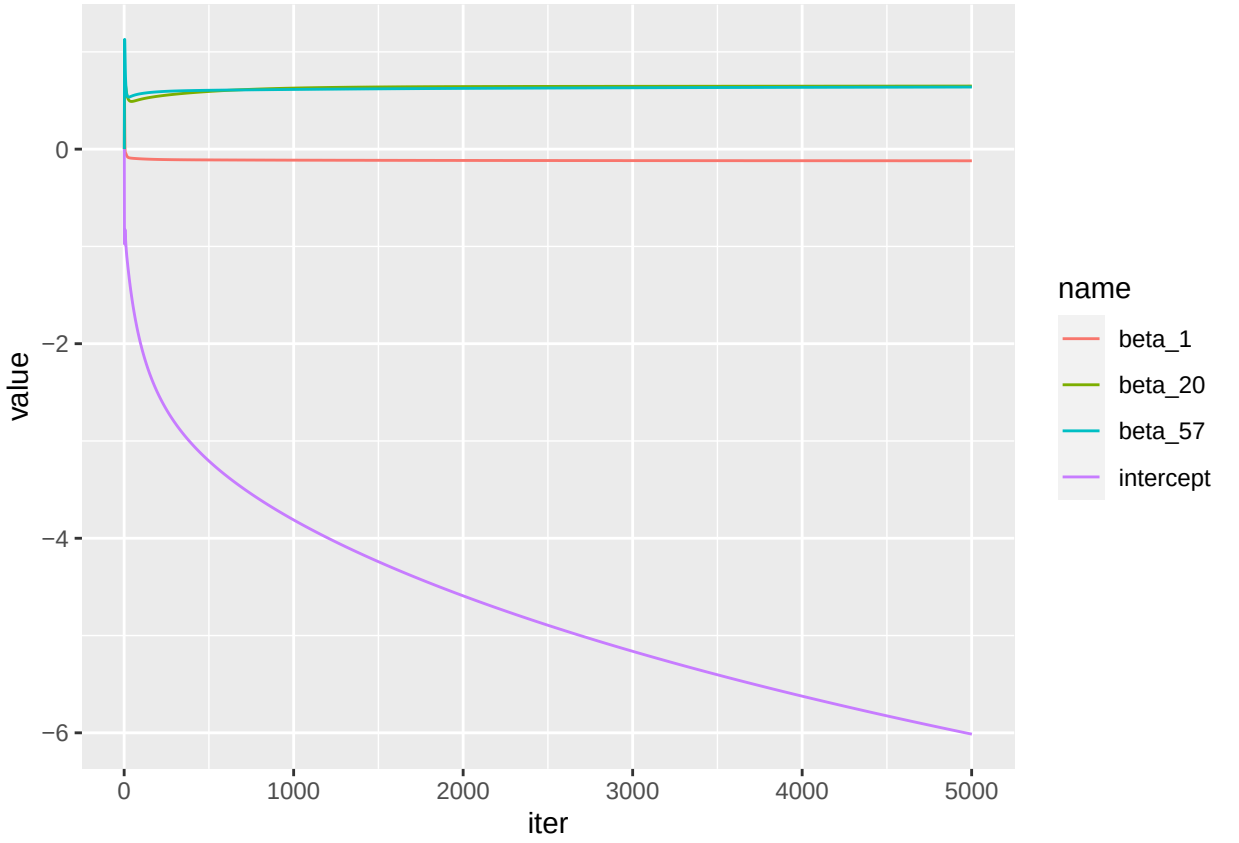
Then, we run using gradient descent

```
t1=Sys.time()
gd_result_sigmoid = solve_gradient_descent(
  X,
  Y,
  link=sigmoid,
  deriv_link=function(x){sigmoid(x) * (1 - sigmoid(x))},
```

```
  alpha=function(i){1e-3},
  # alpha=function(i){
  #   (1e-3) * ((1/2)^floor(i / 10))
  # },
  max_iter=5000,
  tol=1e-5,
  verbose=T
)
t2 = Sys.time()
# save
saveRDS(gd_result_sigmoid, "~/megacron/gd_result_sigmoid.RDS")
### fit the T-model
gd_result_t = solve_gradient_descent(
  X,
  Y,
  link=function(x){pt(x, df=50)},
  deriv_link=function(x){dt(x, df=50)},
  alpha=function(i){1e-4},
  max_iter=750,
  tol=1e-5,
  verbose=T
)
# save
saveRDS(gd_result_t, "~/megacron/gd_result_t.RDS")
```

Sadly, gradient descent takes too long for this dataset (we show it works for a much smaller problem below however). Running the descent for ~ 1.25 hours (5000 iterations), we see that the descent is progressing in the correct direction – it's just getting there really slowly. Notably, starting at small alphas (`1e-3, 1e-4, 1e-5`) and doubling every 10 or 50 steps (as instructed in the problem) resulted in divergence – I found it more stable to keep alpha constant at `1e-3` and `1e-4`. Finally, it should be noted that the intercept appears to be the term reluctant to converge – all other terms tend to match the Fisher result.

```
### collect the history of the descent
gd_sigmoid_history = data.frame(do.call("rbind", gd_result_sigmoid)) %>%
  `colnames<-`(c("intercept", paste0("beta_", 1:57))) %>%
  mutate(iter = row_number())

### plot the slow convergence
ggplot(
  gd_sigmoid_history[, c("iter", "intercept", "beta_1", "beta_20", "beta_57")] %>%
    tidyr::pivot_longer(., cols=c("intercept", "beta_1", "beta_20", "beta_57")),
  aes(x=iter, y=value, color=name)
) + geom_path()
```



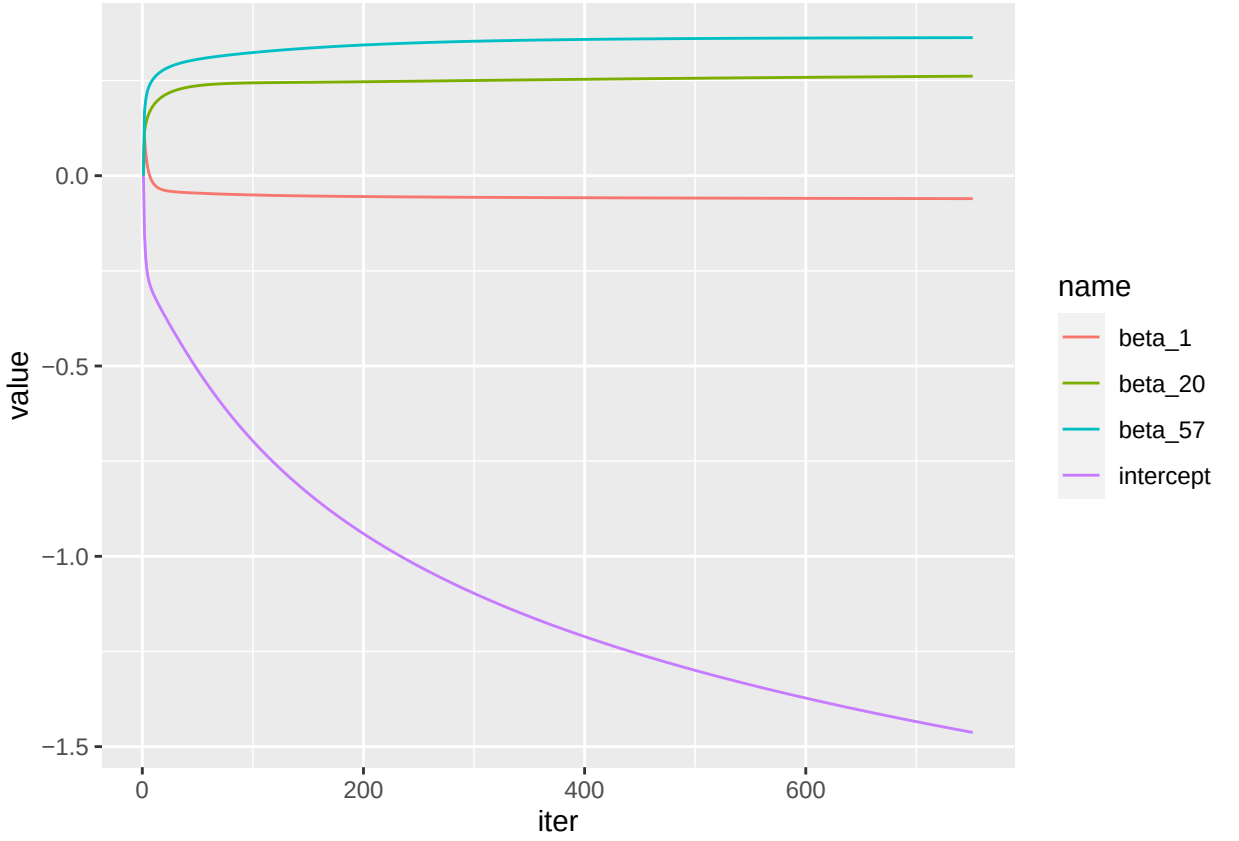

```
### print the coefficients
gd_result_sigmoid[[length(gd_result_sigmoid)]]
```

```
##  [1]  -6.01362919  -0.11959643  -0.24416736   0.04534819   2.53890976
##  [6]   0.55351394   0.35875629   0.94415178   0.35317481   0.29506855
## [11]   0.18298417  -0.01504894  -0.15159340  -0.05157511   0.21360987
## [16]   0.37965003   0.80947341   0.42668707   0.08156031   0.07961197
## [21]   0.64859071   0.30241725   0.16360244   0.78597643   0.64831928
## [26]  -3.03471326  -0.98808012 -17.13442166   0.38754408  -1.30964507
## [31]  -0.17597014  -0.16749004   0.45750890  -0.42235867  -0.24643317
## [36]  -1.18438347   0.40511983  -0.08667248  -0.11727438  -0.33818341
## [41]  -0.09800831  -6.52195871  -1.93576928  -0.31272567  -1.11297011
## [46]  -0.64779500  -1.16844123  -0.21550024  -1.10512222  -0.38958329
## [51]  -0.05514993  -0.03669289   1.01885824   1.25579293   0.80759799
## [56]   0.55764554   0.99937148   0.63857269
```

Lastly, we return to the T-setting, using gradient descent this time as well. It's the same thing: it's getting there, just really slowly. Again, annealing the α hyperparam didn't do much for me.

```
### collect the history of the descent
gd_t_history = data.frame(do.call("rbind", gd_result_t)) %>%
  `colnames<-`(c("intercept", paste0("beta_", 1:57))) %>%
  mutate(iter = row_number())

### plot the slow convergence
```

```
ggplot(
  gd_t_history[, c("iter", "intercept", "beta_1", "beta_20", "beta_57")] %>%
    tidyr::pivot_longer(., cols=c("intercept", "beta_1", "beta_20", "beta_57")),
  aes(x=iter, y=value, color=name)
) + geom_path()
```




```
### print the coefficients
gd_result_t[[length(gd_result_t)]]
```

```
##  [1] -1.462771905 -0.060335113 -0.151237728  0.042149730  0.606700187
##  [6]  0.307882702  0.189208926  0.465026587  0.169524713  0.143202387
## [11]  0.097931679  0.014856202 -0.088773411 -0.030889475  0.127733231
## [16]  0.225703132  0.458601517  0.232559552  0.056215240  0.041844393
## [21]  0.261459946  0.166345480  0.138593021  0.476687884  0.405199481
## [26] -1.268499881 -0.619625432 -2.292581807  0.209822627 -0.619800166
## [31] -0.119967954 -0.158155552 -0.077563503 -0.280211867 -0.266888496
## [36] -0.614596819  0.204772133 -0.061911210 -0.058578192 -0.181755762
## [41] -0.110425581 -0.911616619 -1.001875875 -0.161625941 -0.580663780
## [46] -0.355962340 -0.580282659 -0.104996411 -0.570533974 -0.250708313
## [51] -0.055463662 -0.051330900  0.564551340  0.644148774  0.476130415
## [56]  0.001442928  0.467475994  0.362689181
```

It's the same story here: tiny α are necessary to prevent divergent steps, but this means that it runs infeasibly slow

Proof of Concept

Now, the above runs of GD are infeasibly slow, in large part due to the (i) size and dimension of the dataset and (ii) the fact that a tiny step size is necessary to prevent a divergent step (which induces a 0/1 prediction and breaks the run). To convince you that the code above in fact works, consider the same questions, albeit run on the smaller toy dataset below. Here, we have 1000 randomly generated observations, with a design matrix containing an intercept and 3 random $\sim N(0, 1)$. The toy dataset is constructed as follows:

```
### make toy dataset
S = 1000
set.seed(2020)
Y = rbinom(S, 1, .3)
X = cbind(
  rep(1, S),
  rnorm(S),
  rnorm(S),
  rnorm(S)
)

glm(Y ~ V2 + V3 + V4, data=data.frame(cbind(X, Y)), family="binomial") -> base_fit
```

We first fit the Fisher-scored logistic regression:

```
solve_fisher_scoring(
  X,
  Y,
  link=sigmoid,
  deriv_link=function(x){sigmoid(x) * (1 - sigmoid(x))},
  max_iter=50000,
  tol=1e-5
)
```

```
## ###### Fisher Scoring Complete ######
## Iterations:  5
## Converged:  TRUE
## L2 Norm:  3.218367e-07
## ######################################
```

```
## [1] -0.93390301  0.08203214 -0.01289467  0.02305484
```

This matches the built-in fit identically:

```
base_fit
```

```
##
## Call:  glm(formula = Y ~ V2 + V3 + V4, family = "binomial", data = data.frame(cbind(X,
##     Y)))
##
## Coefficients:
## (Intercept)           V2           V3           V4
##    -0.93390      0.08203     -0.01289      0.02305
##
```

```
## Degrees of Freedom: 999 Total (i.e. Null);  996 Residual
## Null Deviance:       1192
## Residual Deviance: 1190  AIC: 1198
```

Moreover, it also matches the gradient-descent fit:

```
gd_result_sigmoid = solve_gradient_descent(
  X,
  Y,
  link=sigmoid,
  deriv_link=function(x){sigmoid(x) * (1 - sigmoid(x))},
  alpha=function(i){1e-3},
  max_iter=10000,
  tol=1e-5,
  verbose=F
)
```

```
## ###### Gradient Descent Complete ######
## Iterations:  24
## Converged:  TRUE
## L2 Norm:  9.111247e-06
## ########################################
```

```
gd_result_sigmoid[[length(gd_result_sigmoid)]]
```

```
## [1] -0.93389202  0.08202488 -0.01288541  0.02305463
```

For the Fisher-scored T, we then have:

```
solve_fisher_scoring(
  X,
  Y,
  link=function(x){pt(x, df=50)},
  deriv_link=function(x){dt(x, df=50)},
  max_iter=50000,
  tol=1e-5
)
```

```
## ###### Fisher Scoring Complete ######
## Iterations:  5
## Converged:  TRUE
## L2 Norm:  1.898467e-06
## ########################################
```

```
## [1] -0.58015202  0.04934836 -0.00749756  0.01421404
```

Meanwhile the gradient descent fit is

```
gd_result_t = solve_gradient_descent(
  X,
  Y,
  link=function(x){pt(x, df=50)},
  deriv_link=function(x){dt(x, df=50)},
  alpha=function(i){2e-4},
  max_iter=10000,
  tol=1e-5
)
```

```
## ###### Gradient Descent Complete ######
## Iterations:  42
## Converged:  TRUE
## L2 Norm:  9.696082e-06
## ########################################
```

```
### print the coefficients
gd_result_t[[length(gd_result_t)]]
```

```
## [1] -0.58012291  0.04933403 -0.00747662  0.01421390
```

Hence, we see that the functions in principle work – however, size and numeric issues make the full fit difficult on local CPU.