Lab 10: Poisson and Negative Binomial Models: Answers

TA: Eric Parajon

Code has been adapted from Simon Hoellerbauer and Isabel Laterzo.

Load necessary packages:

Poisson

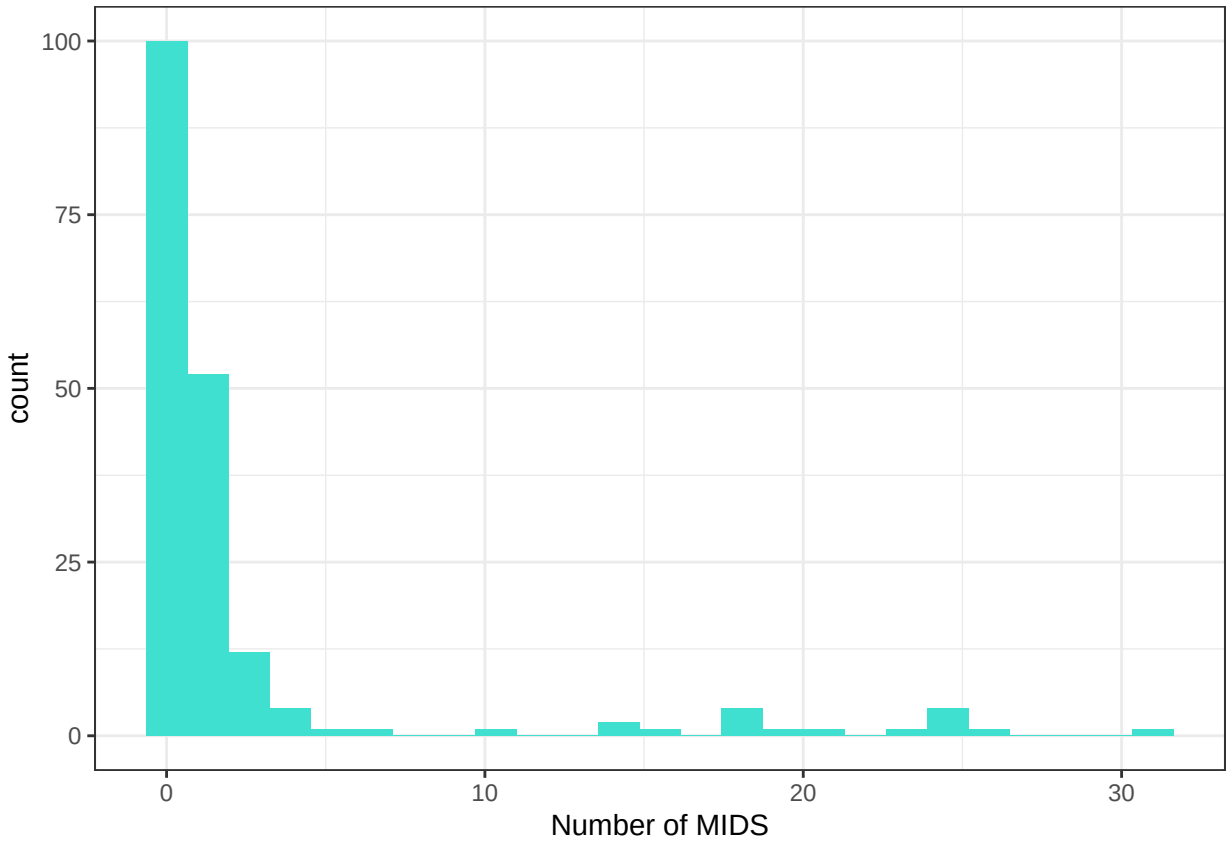
We can apply the types of exercises we've used for logits and multinomial logits to poisson models.

First, get our data. These data are counts from Maoz (2006) of interstate war.

You'll need to change the directory to bring in the file.

We'll be using the variable "nowars20" as our DV - this counts the number of wars in the international system. Here's our histogram:

```
ggplot(data=count, aes(x=nowars20)) +
  geom_histogram(bins=25, fill="turquoise") +
  labs(x="Number of MIDS") +
  theme_bw()
```



This variable is an interesting one in that it's possibly a bit larger (higher range of counts) than many we'd encounter.

Nonetheless, let's build our model. We're going to look at the number of militarized interstate wars based on 'trade network polarization' (lgtrdnpi2) and the number of democratic cliques (clqdemreg) in the system:

```
m1 <- glm(data = count, nowars20 ~ lgtrdnpi2 + clqdemreg, family="poisson")
summary(m1)
```

```
##
## Call:
## glm(formula = nowars20 ~ lgtrdnpi2 + clqdemreg, family = "poisson",
##     data = count)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.6641  -2.3171  -1.5162  -0.9497  10.5572
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   2.7016     0.1482  18.234   <2e-16 ***
## lgtrdnpi2    -9.1513     1.0673  -8.574   <2e-16 ***
## clqdemreg    -5.4620     0.5480  -9.968   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 1182.2  on 126  degrees of freedom
## Residual deviance: 1038.3  on 124  degrees of freedom
##   (60 observations deleted due to missingness)
## AIC: 1222.9
##
## Number of Fisher Scoring iterations: 7
```

Note that this tells us "Dispersion parameter for poisson family taken to be 1."

So, our constant gives the prediction if all X variables at zero (not always substantively meaningful, since some variables are not/cannot be 0).

The coefficient of each x is the expected difference in y (on the logarithmic scale) for each additional unit of the X. To interpret coefficients, we can exponentiate them and treat them as multiplicative effects.

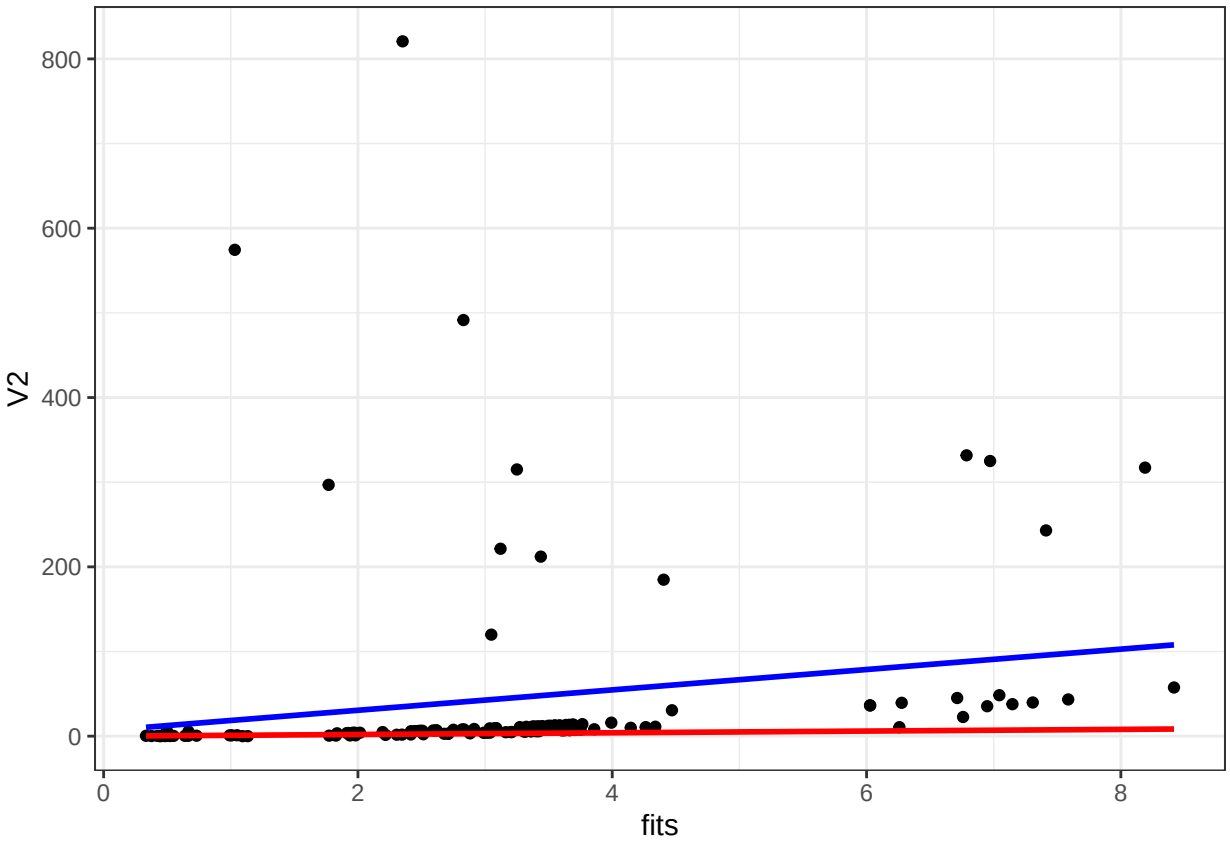
Before we delve further into this model, we might want to check for overdispersion (what's this?).

One approach is to plot the mean of the data against the estimated variance.

```
fits <- fitted(m1) #what does fitted do, do you think?

over <- data.frame(cbind(fits, (model.frame(m1)$nowars20-fits)^2)) #Use squared residuals to approx. va

ggplot(data=over, aes(x=fits, y=V2))+
  geom_point()+
  geom_smooth(aes(x=fits, y=V2), method=lm, se=FALSE, color="blue")+ #line for mean of estimates agains
  geom_smooth(aes(x=fits, y=fits), method=lm, se=FALSE, color='red')+ #assumes mean = variance
  theme_bw()
```




```
  #Add legend
```

We can also check with the dispersiontest() in the AER package:

```
dispersiontest(m1)
```

```
##
##  Overdispersion test
##
## data:  m1
## z = 2.7852, p-value = 0.002674
## alternative hypothesis: true dispersion is greater than 1
## sample estimates:
## dispersion
##   15.70482
```

So, our ALTERNATIVE hypothessis is that the true dispersion is greater than 1. What is our p-value? So what is supported - null or alternative hyptohesis? Do we have overdispersion?

— don't look —- — don't look —- — don't look —- — don't look —- — don't look —- — don't look —- — don't look —- — don't look —- — don't look —- — don't look —- — don't look —- — don't look —- — don't look —-

YES! Looks like we might be dealing with some overdispersion. What can we do to correct this? Let's try a new model - a negative binomial model. While Poisson assumes that the mean and variance are the same, the Negative Binomial model better fits data shows variation which is greater than the mean. The NB model has an additional parameter compared to the Poisson, and this parameter allows us to adjust for more dispersion in the data. In fact, the Poisson model is a special case of the NB.

The glm function does not handle NB models. We now need to use a new function - glm.nb - which is especially for NB models.

```
m2<- glm.nb(data = count, nowars20~lgtrdnpi2+clqdemreg)
summary(m2)
```

```
##
## Call:
## glm.nb(formula = nowars20 ~ lgtrdnpi2 + clqdemreg, data = count,
##     init.theta = 0.2576310461, link = log)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.2711  -1.1263  -0.6884  -0.3945   2.6493
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   2.3463     0.5717   4.104 4.06e-05 ***
## lgtrdnpi2    -6.8139     3.7311  -1.826   0.0678 .
## clqdemreg    -4.4331     1.3877  -3.195   0.0014 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for Negative Binomial(0.2576) family taken to be 1)
##
##     Null deviance: 121.89  on 126  degrees of freedom
## Residual deviance: 111.29  on 124  degrees of freedom
##   (60 observations deleted due to missingness)
## AIC: 476.67
##
## Number of Fisher Scoring iterations: 1
##
##
##               Theta:  0.2576
##           Std. Err.:  0.0428
##
##  2 x log-likelihood:  -468.6720
```

```
coefs<-cbind(coef(m1), coef(m2))
coefs
```

```
##                  [,1]      [,2]
## (Intercept)  2.701618  2.346344
## lgtrdnpi2   -9.151337 -6.813913
## clqdemreg   -5.462049 -4.433117
```

What happened to the coefficients? What happened to the standard errors?

We also can look at AIC - let's compare the two models:

```
AIC(m1)
```

```
## [1] 1222.857
```

```
AIC(m2)
```

```
## [1] 476.6724
```

```
#which is better?
```

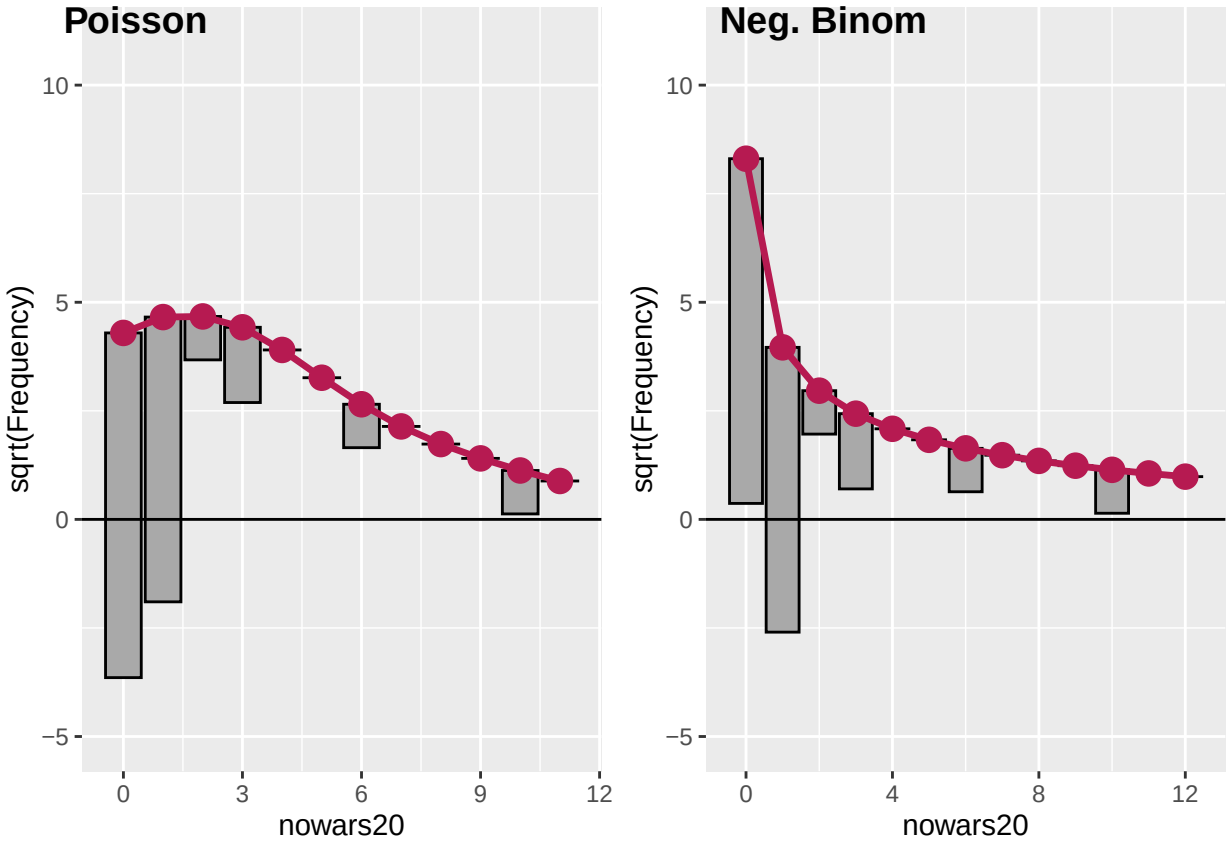
Let's also compare these models using a rootogram:

```
#install.packages("countreg", repos="http://R-Forge.R-project.org")
library(countreg)
library(cowplot)
```

```
## Warning: package 'cowplot' was built under R version 4.2.1
```

```
root.pois <- rootogram(m1, style = "hanging", plot = FALSE)
root.nb <- rootogram(m2, style = "hanging", plot = FALSE)

ylims <- ylim(-5, 11)  # common scale for comparison
plot_grid(autoplot(root.pois) + ylims, autoplot(root.nb) + ylims,
          ncol = 2, labels = c("Poisson", "Neg. Binom"))
```



What does this tell us? The rootogram reflects fitted counts. If a bar doesn't reach the zero line, the model is overpredicting a certain count bin. If it exceeds the zero line, it underpredicts. So, neither of these models are perfect. BUT - the poisson model seems to both over and underpredict a lot. The negative binomial model on the other hand definitely over predicts, but not as much.

#Predicted Counts

For the sake of illustration (and because we see it is a better fitting model), let's use the NB model to plot predicted counts, using simulation.

```
set.seed(8970)
Betas <- m2$coefficients
vcv <- vcov(m2)
```

```r
#Set a number of simulations.
m2_sims<-1000

#Create our matrix of simulated data- we'll limit our dataset to do so.
sims_mat <- mvrnorm(m2_sims, Betas, vcv) #asymptotic sampling distribution of the coefs is the same as
dim(sims_mat)
```

```
## [1] 1000    3
```

```r
temp_data <- model.matrix(m2)

pc_sims <- matrix(NA, nrow = m2_sims, ncol = nrow(temp_data))

#Fill in our expected counts with a for() loop
for(i in 1:m2_sims){
  pc_sims[i, ] <- exp(temp_data%*%sims_mat[i, ])
}


#What if we wanted a single point estimate of our predicted count of wars given our IVs?
mean(pc_sims)
```

```
## [1] 3.111804
```

```r
#And the upper and lower boundaries of a confidence interval based on this simulation?
quantile(pc_sims, prob= 0.025)
```

```
##      2.5%
## 0.4225063
```

```r
quantile(pc_sims, prob= 0.975)
```

```
##    97.5%
## 7.889981
```

How do we predict counts over the range of a variable? Like we always have! Let's say we want to know what happens to the predicted number of counts as the average number of democratic cliques varies from its min to its max.

```r
temp_data <- count

dem_cl <- seq(from = min(count$clqdemreg, na.rm = T),
              to = max(count$clqdemreg, na.rm = T),
              length.out = 50)

pc_dem_cl <- purrr::map_dfr(dem_cl, function(x){

  temp_data$clqdemreg <- x

  pc_df <- exp(model.matrix(m2, data = temp_data) %*% t(sims_mat))

  pc_val <- as.data.frame(t(colMeans(pc_df)))

})

#we could do this in one step, but I like keeping all of the simulated counts so
#we can do whatever we want with the distribution of predicted counts
```

```
plot_data <- t(apply(pc_dem_cl, 1, quantile, probs = c(.1, .5, .9))) %>% as.data.frame()
plot_data$dem_cl <- dem_cl
```

```
#Now to plot:
colnames(plot_data)<-c("low", "med", "high", "Dem_Cliques")

#Collapse information into 3 vectors: for each value of trade in our data,
#give us the average point estimate (expected count), the low bound of a confidence interval, and the h

pc_plot <- ggplot(data=plot_data) +
  labs(x = "Average Number of Democratic Cliques",
       y = "Predicted Count of Intl. Wars") +
  geom_line(aes(x = Dem_Cliques, y = med,
                color = "Number of Interstate Wars"), size = 1) +
  geom_ribbon(aes(x = Dem_Cliques, ymin = low, ymax = high,
                  fill = "80% Confidence Interval (Simulation)"), alpha = .5) +
  theme_bw() +
  theme(legend.title = element_text(),
        legend.position = "bottom", panel.grid.minor.x=element_blank(),
        panel.grid.minor.y = element_blank()) +
  scale_fill_manual(values = "steelblue3", name = "")+
  scale_color_manual(values = "black", name = "")
pc_plot
```