Fitting (Generalized) Linear Models in R

Getting started

Linear models

$$Y_i \sim Normal\left(\mu_i, \sigma^2\right);$$
$$\mu_i = \beta_0 + \boldsymbol{\beta X}_i$$

Generalized linear models

- Logistic regression

$$Y_i \sim Bernoulli(p_i)$$
$$logit(p_i) = \beta_0 + \boldsymbol{\beta X}_i$$

- Poisson regression

$$Y_i \sim Poisson(\lambda_i)$$
$$log(\lambda_i) = \beta_0 + \boldsymbol{\beta X}_i$$

Generating random variables in R

- Create a desktop folder called “Lab 3”
- Open Rstudio, set the working directory to “Lab 3”, open a new script
- Purpose of simulating data: we know the true population parameters, and we can test how well certain analytical models estimate these parameters, under different sample sizes, and so forth
- Start out with a Normally distributed (→ continuous) random variable
- Compare sample mean with true population mean for multiple sample sizes

Generating random variables in R

- Create two numerical variables, mu and sigma, and assign them values of 80 and 10, respectively
- Now you can simulate a sample from the population that is characterized by these true population parameters, using the function rnorm(): r=random + norm=Normal distribution
- Find out which arguments are in the rnorm function
- Now, create a vector called sample1, and draw n=10 samples from Normal(80,10)

```
> sample1<-rnorm(10, mu, sigma)
```

- Note that rnorm() uses the standard deviation (σ), rather than the variance (σ^2)
- Look at the distribution of values in sample1 using a histogram – does the distribution look Normal?

Generating random variables in R

- Now, create 4 more vectors, called sample2 – sample5, with sample sizes 20, 50, 80 and 100
- Look at each sample with a histogram – when do you start to see a Normal pattern?
- Create another vector that holds the sample means for each of the 5 sample vectors
- Remember: the sample mean is an estimate of the population mean
- Calculate the difference between each sample mean and the true population mean – which sample gives you the closest estimate of the population mean?
- People have different answers because R generates different random numbers
- To make sure we all use the same random numbers, we need to set a simulation seed → set.seed(1)

Generating random variables in R

- Let's create a random variable Y that is linearly related to a predictor variable X

→ Generate data simulating a linear regression

- What are the pieces we need?
- Start out by defining the simple numerical variables
- `n<-50 #sample size`
- `beta0<-1.5 #intercept of the regression line`
- `beta1<-0.5 #positive slope; change in Y with unit change in X`
- `error.var<-2 #error variance`
- Next, generate a predictor variable, X
- We will use a Uniform(0,10) distribution to generate X; the R function is runif()
- Use R help to figure out how to generate n values from Uniform(0,10)

Generating random variables in R

- Remember that the relationship with X is modeled for the ***expected value of Y***

$$Y_i \sim Normal(\mu_i, \sigma^2);$$
$$\mu_i = \beta_0 + \boldsymbol{\beta X}_i$$

- So the next step is to generate the expected values based on the linear relationship

```
>exp.v<- ???
```

- Finally, we have the pieces to generate the response variable Y, using rnorm(), the expected values, and the error variance
- Careful: we specified the ***error variance***, but rnorm() uses the ***standard deviation***!

```
>Y<- ???
```

- When you have all the pieces, execute all lines of code starting with set.seed(1), to make sure we all have the same numbers

Generating random variables in R

- Plot Y against X (use the plot() function) – what pattern do you see?
- We have now generated 50 data points (pairs of X and Y)
- Because we simulated these data, we know the true population parameters that generated these observations
- But we can now proceed as if we did not know the true population parameters, and estimates these using a linear regression model
- This is a useful exercise, for example, when you are planning a study and want to understand how many samples you need to collect to achieve a certain level of precision; or to detect an association between X and Y
- It will also give us a chance to see how models are fit in R

Linear models in R

- The function to fit a linear model in R is called lm()
- Use R help to look at the arguments of the lm() function
- In this simple example, we only need to specify the formula
- R formulas have the general form of Y~X, read: “Y is a function of X”
- Technically, in a linear regression, Y is a function of an intercept and X;
- intercepts are indicated with “1” in R formula notation: Y~1+X
- But even if we don’t explicitly include the intercept in the formula, the model will automatically estimate it, so that Y~X is equivalent to Y~1+X
- Create a linear model object

```
> mod1<-lm(Y~X)
```

- Note that you need to use the proper variable names in the formula, which in our case happen to be X and Y

Linear models in R

- Look at the model object mod1:

```
>mod1

Call:
lm(formula = Y ~ X)

Coefficients:
(Intercept)            X
     1.5887         0.5096
```

The model you fit

Parameter estimates

Linear models in R

- More detailed model output:

```
>summary(mod1)
```

…

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   1.5887     0.4118   3.858 0.000341 ***
X             0.5096     0.0690   7.386 1.89e-09 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.315 on 48 degrees of freedom
Multiple R-squared:  0.5319,     Adjusted R-squared:  0.5222
F-statistic: 54.55 on 1 and 48 DF,  p-value: 1.889e-09
```

Linear models in R

- The model object mod1 contains a lot of information
- Use str(mod1) to look at all the pieces of information stored in the object
- It includes predicted values of Y (remember, these are what we expect Y to be based on the regression line, if there was no random variation)
- We could use this information to plot results, but we will learn about the predict.lm() function for that purpose instead
- predict.lm() uses a fitted model and a set of values for the predictor variable to generate predicted values of Y, including confidence intervals for these predicted values
- First, set up a data frame called X.new, with a single column named X, with a sequence of values from 0 to 10, at intervals of 0.2
- Then use the following command, and look at the resulting object

```
> Y.exp<-predict.lm(mod1, X.new, interval="confidence")
```

Linear models in R

```
>head(Y.exp)
  fit       lwr      upr
1 1.588694 0.7606354 2.416753
2 1.690611 0.8872115 2.494011
```

- Plot first column against X-values in X.new using type="l"
- Column 2 and 3 give us the lower and upper confidence interval bound and we can add those to the plot to show how certain we are about the depicted relationship

```
points(X.new$X,Y.exp[,2], type="l",lty=2)
points(X.new$X,Y.exp[,3], type="l",lty=2)
```

- Finally, add observations (X and Y) to the plot as points

Generalized linear models in R

- Poisson regression:

$$Y_i \sim Poisson(\lambda_i)$$
$$log(\lambda_i) = \beta_0 + \beta_1 X_i$$

- Response variable: counts
- Relationship with covariates modeled on the expected value = Poisson mean, using a log-link function
- Using the same sample size, intercept, slope and covariate (n, beta0, beta1, X) as before, generate data from a Poisson distribution using rpois(); call the vector holding the data Y.p
- We can fit a Poisson (or other non-normal) regressions using glm()

```
> mod2<-glm(Y.p~X, family="poisson")
```

- "family" defines the distribution we use to describe our data
- glm(…,family="gaussian") = lm(…)

Generalized linear models in R

- Poisson regression:

```
> summary(mod2)

…

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) 1.500655   0.057686   26.01   <2e-16 ***
X           0.500662   0.007169   69.84   <2e-16 ***
```

Generalized linear models in R

- Poisson regression:
- Use predict.glm() just as you used predict.lm() to generate expected values for Y.p and plot these against X.new$X (ignore confidence intervals for the time being)
- What does that plot look like?
- What are the axis labels?

Generalized linear models in R

- Poisson regression:
- If we want to see the relationship of Y.exp with X on the original scale, we need to ***backtransform***
- Inverse of log() → exp()
- Make plot of ***backtransformed*** values of expected values for Y.p
- Add observations (Y.p, X) to the plot

Generalized linear models in R

- Poisson regression:
- Now that you know what back-transforming does, here is an automatic way to generate predicted values on the original scale

```
> Yp.exp2<-predict.glm(mod2, X.new,type="response",
                       se.fit=TRUE)
```

- type="response" tells R to generate predicted values on the scale of the response variable (type="link" → predicted values on link scale)
- se.fit=TRUE tells R to also calculate standard errors for the backtransformed predicted values
- That is very useful because we cannot just backtransform standard errors, the way we can backtransform parameters!!!
- The predict(…, type="response") function internally uses the appropriate calculations to return correct standard errors on the natural scale

Generalized linear models in R

- Logistic regression:

$$Y_i \sim Bernoulli(p_i)$$
$$logit(p_i) = \beta_0 + \beta_1 X_i$$

- Data *Y* are binary (0 or 1)
- Relationship with predictor variables is modeled on the expected value (success probability p) of the Bernoulli random variable
- We can generate Bernoulli/Binomial random variables in R using rbinom()
- Instead, we will use an existing data set
- From Canvas, get csv file “BirdsBurn.csv” and save it to the folder “Lab 3”
- Read it into R, call the object “birddata”
- Look at birddata – what do the data look like?
- Y: HEWA detected, yes (1) or no (0) → Binary response variable
- X: location burned yes (1) or no (0) → Binary predictor variable

Generalized linear models in R

- Logistic regression:
- Use glm(…, family="binomial") to fit logistic regression model to data, modeling a relationship of bird detections with burn
- Trick: tell R which data frame to look in for the data:
- glm(…, data=birddata, family="binomial")
- Then, use column names in birddata to specify the formula
- When you are done, look at a summary of the model (mod3)

Generalized linear models in R

- Logistic regression:

```
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)    2.2246     0.5263   4.227 2.37e-05 ***
Burned         0.2321     0.7993   0.290    0.772
```

- Parameter estimates given on the logit-scale → we need to backtransform to understand effect on the probability scale
- Parameters tell us:

$$logit(p_i) = 2.2246 + 0.2321 * Burned_i$$

- What is logit(p) in burned plots?
- What is logit(p) in unburned plots?
- Calculate these quantities in R, calling the variables lp.burned and lp.unburned
- To backtransform to the probability scale, calculate the inverse logit:

$$\exp(x)/(\exp(x) + 1)$$

Generalized linear models in R

- Logistic regression:
- You can also use predict.glm() to calculate predicted values on the logit scale and the natural scale
- Create a data.frame called burn.new, with a column named “Burned”, and two entries, 0 and 1

```
> predict.glm(mod3, burn.new, type="link", se.fit=T)
```

- produces expected values on the logit-scale

```
> predict.glm(mod3, burn.new, type="response", se.fit=T)
```

- produces expected values on the response scale

- Save the output of the last command to an object called Yl.p
- How would you plot these results?

Generalized linear models in R

- Logistic regression:
- Some new lower level plot commands

```
> plot(1:2, Yl.p$fit, pch=19, xlim=c(0.5,2.5),
      ylim=c(0.8,1), xlab="Burn category",
      ylab="Probability of occurrence", axes=F)
```

- xlim, ylim defines range of x, y values displayed on plot
- axes=F suppresses plotting axes
- We can then manually add axes which allows us to manipulate tick marks and labels

```
> axis(side=2)
> axis(side=1, at=c(1,2), label=c("Unburned","Burned"))
> box()
```

- This last command draws a box around the plot

Generalized linear models in R

- Logistic regression:
- Now we can add error bars to the plot, using the arrows() function
- First, we calculate the upper and lower Y values for each error bar
- Estimate + SE; estimate – SE

```
> eup<-Yl.p$fit+Yl.p$se.fit
> elw<-Yl.p$fit-Yl.p$se.fit
```

- Then we use these values to add arrows (ie, error bars)

```
> arrows(x0=1:2, y0=elw, y1=eup, angle=90, code=3)
```

- x0, y0, y1 are coordinates for error bars, angle is the angle between the two lines comprising each error bar, code=3 means, draw a perpendicular line at either end of the vertical line