Cactus Conundrum [Logistic Regression in R]

# Import Data

The data for this assignment is called **ohcrapcacti.csv**. Download it, and import it into R. I’m calling it cactus, so if you don’t want to do any editing just call it that for the rest of the walkthrough.

cactus <- read.csv("ohcrapcacti.csv")

# Packages

They are basically expansion packs for R! We will use two very very common ones - **ggplot2** and **dplyr**, and two that will help us with our confusion later on, called **caret** and **e1071**. To get these expansion packs (only have to do this once unless your computer explodes or something), use the *install.packages()* function like so:

install.packages("ggplot2")  
install.packages("dplyr")  
install.packages("caret")  
install.packages("e1071")

Both have dumb spelling and no capitals, so pay attention and put them into quotation marks. If you get a dumb RTools error, you spelled it wrong.

Now load the packages. They’re installed but you can have so many packages that it’s important to remind R each time which one you’re looking at - like pulling down books before you begin to study, it’s chaos if all the books are out all the time. Use the *library()* function for that.

library(dplyr)  
library(ggplot2)  
library(caret)  
library(e1071)

Fun Fact: if you get a red warning from dply about something something masking something, don’t worry about it! It’s putting the dplyr book of knowledge on top of any other book of knowledge you have loaded - mostly, the regular basic model of R. So that’s fine! Caret also sometimes says it’s overwriting things, that’s fine.

# Renaming Terrible Variables

I like to have incredibly long variable names but they are a pain in the butt. I also like to have clear data, so I call things “yes” and “no” when we need them to be 0 and 1 for our model. We will use the *rename()* and *mutate()* functions from dplyr.

## Renaming

First, wtf are the names?

colnames(cactus)

Wow, they are dumb! Let’s call them **cactus, slope, distance**. Separate each new name with a comma - you can put them on different lines like I have if you enjoy tidy code.

For dplyr, the pipe function (*%>%*) tells R to keep reading. Put it at the end of a function as needed.

cactus <- cactus %>%  
 rename(cactus = Was.There.A.Cactus,   
 slope = Slope.Degree,   
 distance = Minimum.Distance.From.Another.Cactus..m.)

## Mutating

Mutating is changing things. We’ll also use my favorite function, the *ifelse()* function. Essentially this says we’re going to name a column **ouch**. If there wasn’t a cactus, it’s a 0; if there is a cactus, it’s a 1.

cactus <- cactus %>%  
 mutate(ouch = ifelse(cactus == "No", 0, 1))

You don’t HAVE to do this step, but I’ve seen too many people not realize which one is 0 and which one is 1, and that screws up how you interpret it later so I like it manually. It also helps for ggplot code later on, and a few other things. Manually change your variables to 0’s and 1’s friends! It’s just a better way of being.

# Look At It

I like the ggplot2 package because it makes vizualization of complicated datasets really fast and easy. Let’s look at our two x variables, slope and distance. We’ll use the cactus column to color our data.

ggplot(cactus, aes(x = slope, y = distance, col = cactus))+  
 geom\_point()

Slope and distance have a relationship, but it’s different between the spots with cacti and the spots without.

# Hypotheses

Here are the basic ideas we’re testing in this fake dataset:

* If you are more likely to have a cactus if you are closer to another cactus, then the minimum distance from another cactus should be significant.
* If you are more likely to have a cactus if you are on a steep slope, the increased slope degrees should be significant.

And by significant, we mean significant in a logistic regression!

# Logistic Regression

Let’s start with slope! Use the *glm()* function to build a logistic regression model where **ouch** is the variable we are predicting and **slope** is the predictor. *glm()* only builds the model - use summary() to evaluate it.

slope.model <- glm(ouch~slope, data =cactus, family=binomial)  
summary(slope.model)

Let’s build a simple distance model now, same idea:

distance.model <- glm(ouch~distance, data =cactus, family=binomial)  
summary(distance.model)

Both have significant slopes (0.0170 and 0.00469 respectively), but the distance model has a lower AIC (22.01) which means it is probably a better model. But to double check, you should always look at accuracy!

# Accuracy

We’ve built these fancy models, so let’s see how well they work. First, tell them to take the data and predict what it should be. It spits out probabilities which is annoying so use the *round()* function to make it into a 0 or a 1.

cactus$slope.predict <- round(predict(slope.model, newdata=cactus, type='response'),0)  
cactus$distance.predict <- round(predict(distance.model, newdata=cactus, type='response'),0)

Equally annoying, the code we’re going to use now doesn’t like it if our numbers aren’t factors. Long story. Also ggplot doesn’t like ’em so we’re adding a 1 to the end of the names. Use the *as.factor()* and *mutate()* code to make them factors.

cactus <- cactus %>%  
 mutate(ouch1 = as.factor(ouch)) %>%  
 mutate(slope.predict1 = as.factor(slope.predict)) %>%  
 mutate(distance.predict1 = as.factor(distance.predict))

OK, now you can use the *confusionMatrix()* function. The data argument is the real answer, and reference is what the model guessed.

confusionMatrix(data = cactus$ouch1, reference= cactus$slope.predict1)  
confusionMatrix(data = cactus$ouch1, reference= cactus$distance.predict1)

Two important bits for reading:

* Accuracy is a percent - so 74% for the slope model, and 88% for distance
* Sometimes models like to predict everything is one category. So look at the ‘sensitivity and specificity’ section - if one of those is realllllly low, that’s bad.

For example, if the slope model predicted everything was a 1 (aka welcome to cactus town), then specificity would be 100%, but sensitivity would be 0%.

# Visualizing

OK, let’s take one last look at our models using ggplot. The *geom\_smooth()* function here is really long, but it shows you what the logistic regression looks like. You want that to look like a fancy S (or reverse S), but where at no point you could just draw a flat line through the grey cloud. That would mean there was no real relationship.

ggplot(cactus, aes(x = slope, y = ouch))+  
 geom\_point( col = 'blue')+  
 geom\_smooth(method='glm', method.args=list(family='binomial'))

ggplot(cactus, aes(x = distance, y = ouch))+  
 geom\_point( col = 'red')+  
 geom\_smooth(method='glm', method.args=list(family='binomial'))

Both of them are kind of curving, but you can see that the distance model has a much nicer curve to it. This is because there is less overlap between ‘yes there is a cactus’ and ‘no there isn’t’ than if you look at slope.

For our purposes that’s it! You can built multivariate versions but this model is great with just distance, so why over-complicate it?