donner.r: Explanation of code

Goals of code:

- Fitting a logistic regression with a 0/1 response
- Fitting a logistic regression with a factor variable as response
- Fitting a logistic regression with a factor predictor

The data are in donner.csv. The two character variables, femc and survivalc, are converted to factors.

Fitting a logistic regression: `glm(, family=binomial)`
A logistic regression is fit using the glm() function, which fits a variety of generalized linear models. The formula is familiar: response $\sim$ X variables. The data= argument is familar. The new, **and necessary pieces**, are glm() not lm() and family=binomial. The family= argument tells glm() about the error distribution for the response. For logistic regression, you want binomial. If you omit family=, the default is a gaussian distribution, which is the same as using lm(), even though you request glm().

Take home: remember family= whenever you use glm().

The quantity being modeled is the log odds of a 1. Since you can choose which outcome is labelled 0 and which is labelled 1, you can choose which event is being modeled (i.e., make the desired event have the value 1). In the Donner party data set, survive = 1 means an individual survived, the quantity being modeled is the log odds of survival. Backtransforming this to a probability gives you the probability of survival. If you redefined death = 1, then you would model the log odds of death and the backtransformation gives you the probability of dying.

All the usual helper functions, coef(), summary(), predict(), resid(), AIC(), BIC() do the appropriate things for a glm() fit. The base R anova() function gives you sequential tests of model terms. I (and most others in the US) prefer partial tests almost all the time. For models with only factor variables, `joint_tests()` in the emmeans library is glm()-aware and gives you partial tests.

The output from summary() includes a summary of the deviance residuals (a type of residual that makes sense for Yes/No or count data), a table of regression coefficients with se's and tests of coefficient = 0, two deviance values and the AIC statistic for the fitted model. The only really new concept are the two deviance values. The value labeled Null deviance is the deviance for the intercept-only model (i.e. without any model term). The df for this is N-1, where N is the number of observations. The value labeled Residual is the deviance for the fitted model. This is N-(k+1) where k is the number of variables in the regression model.

Hopefully, there is a large change in the deviance when you add all the model terms. You can turn

the change into a test by fitting the null model and using the null and the full model (i.e., two specific models) in anova(). This is illustrated a little later in the code.

Plotting predicted values on a grid of potential X variables: `expand.grid()`
predict() gives you predicted values for the observations used to fit the model. To get predicted values for new observations, you need to provide their X values. To get predicted values for a range of observations, e.g. ages 15, 16, $\cdots$ 65 for fem=0 (males) and fem=1 (females), you need a grid of all combinations of age and fem. expand.grid() is a handy tool for generating such a grid. expand.grid(a = vector, b = vector) gives you two columns, labeled a and b, with all combinations of the a vector and the b vector. The head and the tail commands show you the first 6 and the last 6 rows of donner.new.

You can get predictions for the new X's the same way as you get predictions for new X's for an lm() fit, using predict(model, newdata=).

Predictions from a glm() object: `predict(, type=)`
There are two commonly-used types of predictions from a glm fit. One is the linear predictor: $\beta_0 + \beta_1 X_1 + \cdots$. For a logistic regression model, this predicts the log odds for an observation. The second is the predicted probability (the response). This is the value of the linear predictor back-transformed to the response scale. Both have their uses.

The linear predictor is the default. To get the predicted probabilities, add `type='response'` to predict().

Plotting multiple lines on one graph:
There are various ways to do this. If you use ggplot graphics, that will be done completely differently. I show one way to do this using base graphics. matplot() plots matrices as different symbols or different color lines. The only trick is that the vector with predictions for both fem=0 and fem=1 needs to be reshaped into a matrix with one column for fem=0 and one column for fem=1. That's what the matrix() command does. The number of rows is the number of age values.

Testing comparisons between specific models: `anova(, test='Chi')`
anova() allows you do test specific pairs of models. The code illustrates the comparison of the logistic regression to an intercept only model. You fit both models, then use anova(reduced model, full model), where the larger model is second.

By default, anova() only gives you the change in deviance. To get the p-value associated with this change in deviance, add `test='Chi'` to get the usual test.

Fitting a logistic regression with a factor variable as response: `glm(survivalc ~ )`
The previous illustrations of glm() used a response variable with values of 0 or 1. glm() modeled the probability of a 1. You can also use a factor as a response. survivalc is the factor version of "No" or "Yes" for survived or not. That can be used as the response variable. The event that is modeled is the "last" value in alphabetic order. Or if you print the levels of the factor, using `levels(donner$survivalc)`, the event is the last level in the vector of levels. For survivalc, that is

Yes, so the model with the factor response gives identical results to the model using survival (where 1 = survived).

Fitting a logistic regression with a factor predictor

There are some advantages to letting R create the automatic indicator variables from a factor. When you use a factor, emmeans knows that that variable indicates groups. When fem = 0 or 1, emmeans doesn't know whether that is two levels of continuous variable or groups. `donner$femc` is the factor version of M or F. When you fit a model with femc, emmeans will give you the mean response for M at the average age and for F at the average age. Again, you can get estimates of the linear predictor or of the backtransformed probability.

The coefficient for femc is the negative of that of fem. That's because femc (the factor version of F or M) sets the first level to 0. That makes M have the 1. You see that in the summary() or coef() output. femc is labelled femcM to indicate that is the estimate for femc = M.