Logistic and Multinomial Regression

Irfan Kanat

September 15, 2017

Quick note before we start. This is the first module we use the caret package. The package itself is quite large. So I recommend you install it before you do anything else.

In the previous module we learned about regression, with a continuous dependent variable. In this learning activity we will extend the regression to categorical variables[1]. We will start with case of binary variables, and expand upon it.

What is the Big Idea?

Let us start with a brief discussion of why we can not simply fit an OLS regression to these variables[2].

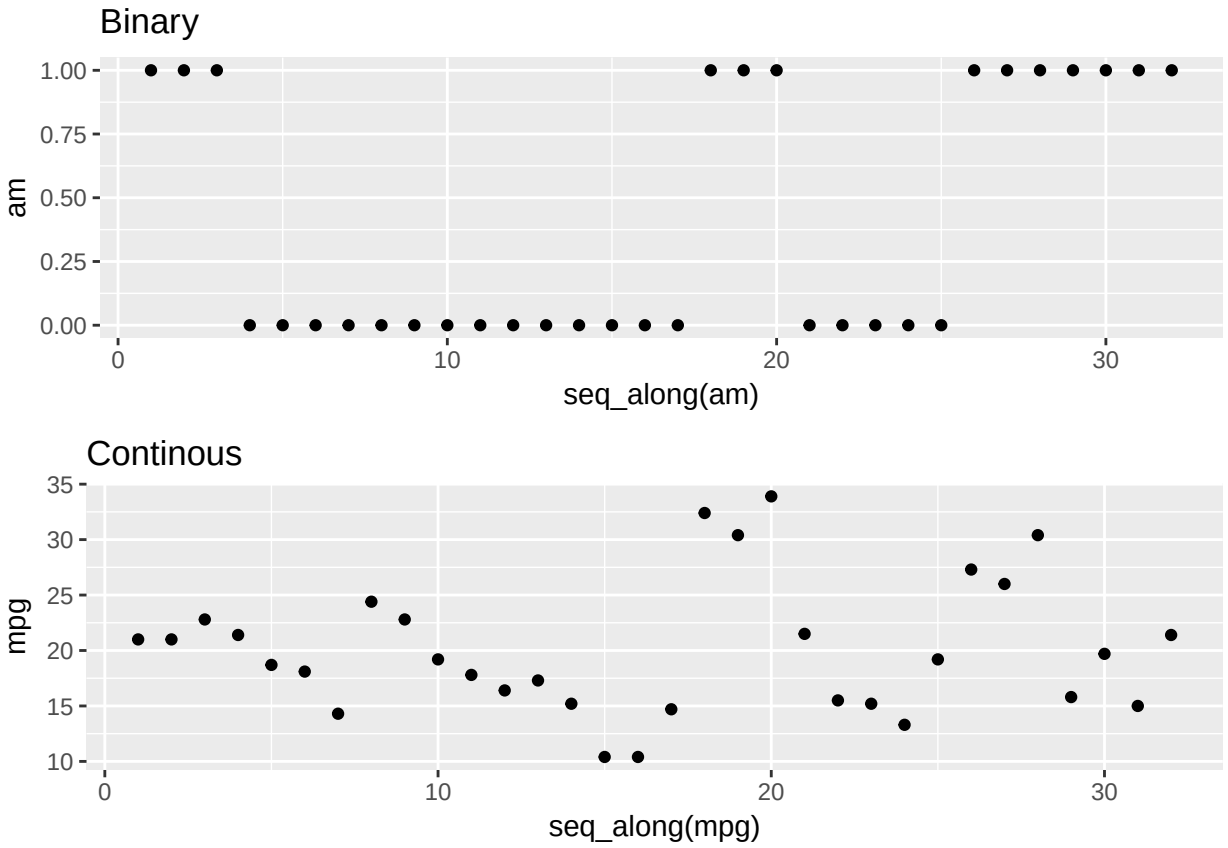
Let us assume a binary variable such as gender. The variable can take two levels: male or female. If you compare this to the continuous variables we used in OLS, you will quickly realize that things are a bit different. Let us look at how binary variables look compared to continuous variables.

I will use the now familiar mtcars dataset. We have used this dataset since the first module, so you should recognize it. I am plotting the type of transmission (Automatic/Manual) and miles per galon. The transmission type is binary.

```
plotContinuous <- qplot(x = seq_along(mpg), y = mpg, data = mtcars) + ggtitle("Continous")
plotBinary <- qplot(x = seq_along(am), y = am, data = mtcars) + ggtitle("Binary")
grid.arrange(plotBinary, plotContinuous, ncol=1)
```

[1] A categorical variable is one that takes discrete levels. Gender, Eye Color, Transmission Type (automatic = 0 or manual = 1), or Neighborhood can be considered categorical. We call categorical variables with only two levels as binary variables. Transmission type or pyhsiological gender can be considered binary.

[2] In fact, we can fit OLS to binary variables, these models are called continuous probability models but their use is not very common.



Despite the difference in the scale of the variables, it is clear that there is a lot more levels a continuous variable can take.

The problem is evident in the distribution of binary variable. First, a binary variable is restricted to two levels. When you try to use OLS, you will start violating several assumptions. The regression can produce values greater than 1 and smaller than 0. Second, often times the linearity assumption of OLS is too restrictive in binary dependent variables. Third, OLS assumes error term is normally distributed. With a Binary dependent variable that is hard to satisfy. Finally, OLS assumes homoskedasticty (constant variance) in error terms. Again, this assumption is near impossible to satisfy with a binary variable.

We need to find a way to model the relationship between dependent variable and independent variables without getting caught up with all the above restrictions.

Logistic Regression

So the problem is that the dependent variable is bounded and relation between dependent variable and independent variable may be non-linear. One solution is quite simple. Run your OLS and then carry out a non-linear transformation on the results to conform to the boundaries. This is referred to as a link function. The most popular link function for binary variables is the logit transformation.

Logistic regression is the name given to a regression with logit link function?

How Does Logit Fit in Logistic Regression

With binary dependent variables, what we do is model the probability of membership in a category. This is sensible as the probability ranges from 0-1 in a continuous fashion, whereas the dependent variable is discrete

(0 or 1).

We can use OLS to estimate the probability, and then transform this estimate to conform to restrictions listed above. The formula for logistic regression is given below:

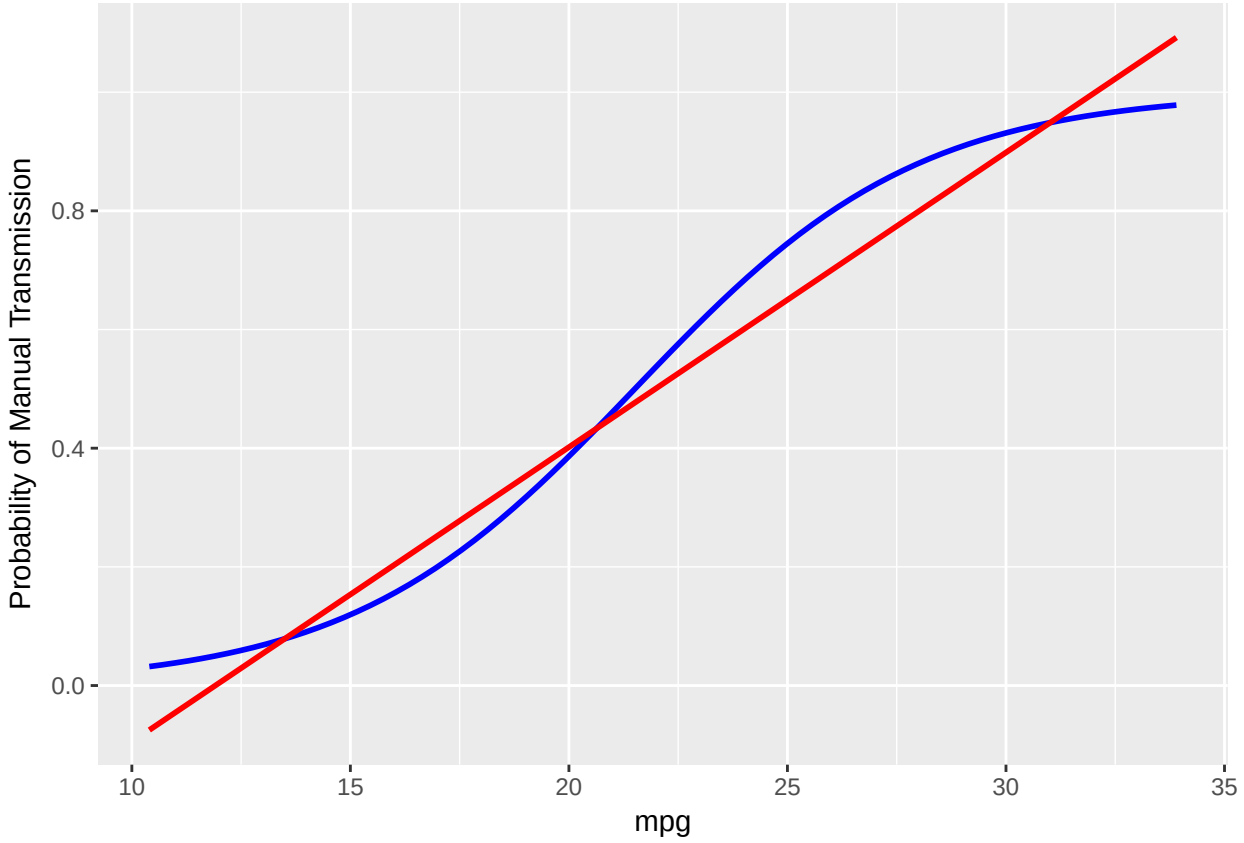
$log(\frac{p}{1-p}) = \beta_0 + \beta_1 \times X_1 + \beta_2 \times X_2$

With two little changes to OLS formulation, we have the logit regression: (1) on the left hand side we have the link function applied to probabilities, (2) there is no error term (probability incorporates error).

How Does it Look

A simplified example with one independent variable (mpg) provides a visual key into what we described above. Compare the blue line below to what an OLS would have provided (red). That is the S shape the logit is famous for. It conforms to the bounds and non-linearity we discussed above.

```
ggplot(mtcars, aes(x = mpg, y = am)) +
    geom_smooth(method="glm", method.args = list(family="binomial"), se=FALSE, color="blue")+
    geom_smooth(method="lm", se=F,  color="red") +
# Bonus: rename the y axis label
        ylab('Probability of Manual Transmission')
```



What is This About Odds, Log Odds and Probabilities?

When you are dealing with logit, you will quickly become familiar with odds, log odds and probabilities. Let me provide a quick overview.

The probability is restricted between 0 and 1... For example, a ten sided die (d10) has 60% to produce a number smaller than or equal to six.

The odds do not have an upper limit and they are non-linear. The odds of obtaining a number smaller than or equal to six on a d10 is 1.5 (p/(1-p) = .6 / .4).

When you take natural logarithm of odds (logit), the lower limit is removed. The log odds of obtaining a number smaller than or equal to six on a d10 is .4 (log(1.5)).

Here is a table to show you how these things work.

```
# For a sequence of probabilites from .1 to .9
p <- seq(.1, .9, by = .1)
# Calculate the Odds
odds <- p/(1-p)
# Calculate the log odds (logit)
lnOdds <- log(odds)
# Present results
data.frame(cbind(p, odds, lnOdds))
```

```
##     p      odds     lnOdds
## 1 0.1 0.1111111 -2.1972246
## 2 0.2 0.2500000 -1.3862944
## 3 0.3 0.4285714 -0.8472979
## 4 0.4 0.6666667 -0.4054651
## 5 0.5 1.0000000  0.0000000
## 6 0.6 1.5000000  0.4054651
## 7 0.7 2.3333333  0.8472979
## 8 0.8 4.0000000  1.3862944
## 9 0.9 9.0000000  2.1972246
```

Note that for probability of .5 the odds are 1 and the log odds are 0.

Inspecting the table, you can see that this type of transformation takes care of boundaries (0 and 1) and linearity. With this transformation, we no longer have to worry about violating the OLS assumptions.

Still, we may need to transform logit results to things more understandable to human beings.

To obtain odds from logit, we exponentiate (reversing the log) both sides.

$\frac{p}{1-p} = e^{\beta_0+\beta_1\times X_1+\beta_2\times X_2}$

From odd's we can move onto probabilities like so:

$p = \frac{1}{1+e^{\beta_0+\beta_1\times X_1+\beta_2\times X_2}}$

Here is a simple exercise for you. Probability of landing a critical hit (rolling a 20 on a 20 sided die) is 5%. Calculate the odds and log odds of critical hit.

How about the probability and odds of getting hit by a fireball, if the log odds are .5?

What is the Catch?

Nothing in life is free... Once you use a link function, the β coefficients are no longer as easy to interpret. We will see how this translate into interpretation of coefficients later on.

Logit in R

We have gone through quite a dry description on why we have logit regression. Let us go over an example to see how all the above discussion fits into practice.

Fit a Logit Model

Let us fit a logit model to predict transmission type of cars in mtcars dataset.

We use glm() to fit these kinds of **g**eneralized **l**inear **m**odels. Please skim the manual page for glm (?glm) before virtual office hours.

```
mtcars_lgt_0 <- glm(am ~ mpg + vs , mtcars, family = "binomial")
```

Let us see what we have done thus far:

1. mtcars_lgt_0 <-: assigns the estimation results into a variable named mtcars_lgt_0.
2. glm(am ~ mpg + vs, mtcars, family = "binomial"): calls lm function with three parameters
   - am ~ mpg + vs : is the formula specification, left of ~ is the dependent variable, and right of ~ is the independent variables.
   - Since am = 1 when car is manual, we model the car having a manual transmission.
   - data = mtcars: name of the dataset to draw the variables from ("data =" ommitted).
   - family = "binomial": Tells R to use logit link function

Evaluate Results

Let us view the results of estimation.

```
summary(mtcars_lgt_0)
```

```
##
## Call:
## glm(formula = am ~ mpg + vs, family = "binomial", data = mtcars)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.3389  -0.6304  -0.2980   0.3069   2.0106
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -9.9183     3.4942  -2.839  0.00453 **
## mpg           0.5359     0.1967   2.724  0.00644 **
## vs           -2.7957     1.4723  -1.899  0.05758 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 43.230  on 31  degrees of freedom
## Residual deviance: 24.944  on 29  degrees of freedom
## AIC: 30.944
##
## Number of Fisher Scoring iterations: 6
```

As with OLS, the output states what the function call was. This is very useful if you somehow lost the code you used to create estimation.

Then comes the deviance residuals. This is an overview of model fit. Do not worry about these, as we will discuss about model fit below.

Coefficients is what most people would be interested in. It shows the estimates for coefficients, we can see miles per galon is a statistically significant predictor of transmission type. The estimate reported here is the effect on log odds.

Below this all is the fit indices.

Evaluate Model Fit

AIC: Among the values reported, the easiest to interpret is AIC. Aikaike's Information Criterion reports the model fit, lower the value, better the fit. It is useful to compare to models fit to the same dataset (not necessarily nested). It's usefulness however is limited to model comparison.

Log Likelihood Test: If you want to make sure the model is actually good, one way is to compare it to a null model (nothing on the right hand side). We can use the null deviance and residual deviance for this purpose. We will compare the fit of our model to see if the model is an improvement over a null model. We will use a chi squared (χ^2) test for this comparison.

If you inspect the summary output, below the coefficients, you will see some deviances reported. We can run a chi square test these values. You will note that the null deviance is 43.2297333 with 31 degrees of freedom. The residual deviance of our model on the other hand is 24.9438078 with 29 degrees of freedom. For chi squared (χ^2) test we will use the difference in deviance with the difference in degrees of freedom. This will tell us if the model is a significant improvement over a null model.

Looking at the results the values are: deviance = 43.2297333 - 24.9438078 = 18.2859255 with df = 31 - 29 = 2. We can then estimate the significance by pchisq(18.2859255, 2, lower.tail = F)

```
# Instead of manually calculating the values, we can also ask R to do it for us
pchisq(mtcars_lgt_0$null.deviance - mtcars_lgt_0$deviance,
       mtcars_lgt_0$df.null - mtcars_lgt_0$df.residual,
       lower.tail = F)
```

```
## [1] 0.0001069699
```

P value smaller than .05 means the model is statistically significantly better than null model.

Evaluate Coefficients

The beta coefficients R reports are the effect on log odds. These do not have an intuitive interpretation. Best we can say is that each additional mile per galon increases the logged odds of the car being a manual by 0.54.

If you want odds-ratios, you will need to exponentiate the coefficients. Let us compare raw coefficients with odds-ratios:

```
mtcars_lgt_0$coefficients
```

```
## (Intercept)         mpg          vs
##  -9.9182931   0.5359476  -2.7956898
```

```
# Exponentiate the coefficients
exp(mtcars_lgt_0$coefficients)
```

```
##  (Intercept)          mpg           vs
## 4.926517e-05 1.709067e+00 6.107273e-02
```

The interpretation of odds-ratios are more straightforward. For one unit increase in mpg, odds of being a manual transmission increase by a ***factor*** of 1.71.

Since vs is insignificant, we should not interpret it. If it were significant, we would interpret it like any other binary independent variable, but the coefficient would be multiplicative in odds-ratio.

While the output has the intercept, it is not meaningful to interpret.

Model Performance in Prediction

Logit model is also used for predictions. How do we go from what we have done so far to predicting the outcome?

We can obtain predicted probabilities from the model with predict function.

```
# Type parameter specifies probability
predict(mtcars_lgt_0, type="response")
```

```
##           Mazda RX4       Mazda RX4 Wag          Datsun 710
##          0.79193126          0.79193126          0.37886276
##      Hornet 4 Drive   Hornet Sportabout             Valiant
##          0.22361814          0.52595831          0.04682721
##          Duster 360           Merc 240D            Merc 230
##          0.09498413          0.58979987          0.37886276
##            Merc 280           Merc 280C          Merc 450SE
##          0.08137654          0.04015149          0.24439135
##          Merc 450SL         Merc 450SLC  Cadillac Fleetwood
##          0.34380126          0.14530746          0.01281251
## Lincoln Continental   Chrysler Imperial            Fiat 128
##          0.01281251          0.11508062          0.99053570
##         Honda Civic      Toyota Corolla       Toyota Corona
##          0.97284923          0.99574179          0.23306049
##    Dodge Challenger         AMC Javelin          Camaro Z28
##          0.16643517          0.14530746          0.05785658
##    Pontiac Firebird           Fiat X1-9       Porsche 914-2
##          0.59191812          0.87184667          0.98230021
##        Lotus Europa      Ford Pantera L        Ferrari Dino
##          0.97284923          0.18995212          0.65472503
##       Maserati Bora          Volvo 142E
##          0.13249465          0.22361814
```

If we know nothing more about the data we can use .5 as a cut-off and say any car that has a predicted probability greater than .6 is a manual. **If we knew the prevalence of manual transmission, we can make a more informed decision.** If the sample we have is representative of the population, then we can use the prevalence of manual transmission in cars as an indicator.

```
mean(mtcars$am)
```

```
## [1] 0.40625
```

We know that roughly 0.41 of cars are manual. That means only 40 percent of our predictions should result in a positive prediction. That means we should set our cut-off at .6 (1-.4).

Let us add two variables to the data.frame and go from there.

```
# Save predicted probabilities
mtcars$prob <- predict(mtcars_lgt_0, type = "response")
# Create a variable for predicted transmission type
```

```r
mtcars$predAM <- 0
# Filter the cars with predicted probabilities greater than .6
# and change their prediction to manual.
mtcars[mtcars$prob > .6, 'predAM'] <- 1
```

Now we can compare what we predicted to what was actually observed.

```r
mtcars[,c("predAM", "am")]
```

```
##                     predAM am
## Mazda RX4                1  1
## Mazda RX4 Wag            1  1
## Datsun 710               0  1
## Hornet 4 Drive           0  0
## Hornet Sportabout        0  0
## Valiant                  0  0
## Duster 360               0  0
## Merc 240D                0  0
## Merc 230                 0  0
## Merc 280                 0  0
## Merc 280C                0  0
## Merc 450SE               0  0
## Merc 450SL               0  0
## Merc 450SLC              0  0
## Cadillac Fleetwood       0  0
## Lincoln Continental      0  0
## Chrysler Imperial        0  0
## Fiat 128                 1  1
## Honda Civic              1  1
## Toyota Corolla           1  1
## Toyota Corona            0  0
## Dodge Challenger         0  0
## AMC Javelin              0  0
## Camaro Z28               0  0
## Pontiac Firebird         0  0
## Fiat X1-9                1  1
## Porsche 914-2            1  1
## Lotus Europa             1  1
## Ford Pantera L           0  1
## Ferrari Dino             1  1
## Maserati Bora            0  1
## Volvo 142E               0  1
```

If the model was perfect, the two columns would match perfectly. Realistically, we can not go through every line and compare the results... There needs to be a better way to find out model performance.

Confusion Matrix

One way to learn about model's predictive performance is a confusion matrix. Easiest way to obtain a confusion matrix is to tabulate predicted and actual category membership data.

```r
table(mtcars[, c("predAM", "am")])
```

```
##       am
## predAM  0  1
```

```
##      0 19  4
##      1  0  9
```

There are four cells here:

True Positives: We predicted 1, and the actual value was 1. True Negatives: We predicted 0, and the actual value was 0. False Positives: We predicted 1, when the actual value was 0. False Negatives: We predicted 0, when the actual value was 1.

From these information we can calculate various metrics of performance. I am listing the commonly used ones below.

Prevalence: What is the prevalence of positives in sample? ((TP+FN)/Total)

Accuracy: How often were we able to predict correctly? ((TP+TN)/Total)

Specificity: How good is our model at predicting negatives? (TN/(TN+FP))

Sensitivity: How good is our model at predicting positives? (TP/(TP+FN))

Positive Prediction Rate: How good is our model in predicting positives, taking into account prevalence. $(sensitivity \times prevalence)/((sensitivity \times prevalence) + ((1 - specificity) \times (1 - prevalence)))$

Negative Prediction rate: How good is our model in predicting negatives, taking into account prevalence. $(specificity \times prevalence)/((specificity \times prevalence) + ((1 - specificity) \times (1 - prevalence)))$

Kappa: How well the classifier performed compared to random prediction.

Read the confusionMatrix manual for more on these measures.

```
# Here we use confusionMatrix function from caret package
confusionMatrix(table(mtcars[,c("predAM", "am")]))
```

```
## Confusion Matrix and Statistics
##
##       am
## predAM  0  1
##      0 19  4
##      1  0  9
##
##                Accuracy : 0.875
##                  95% CI : (0.7101, 0.9649)
##     No Information Rate : 0.5938
##     P-Value [Acc > NIR] : 0.0005536
##
##                   Kappa : 0.7277
##  Mcnemar's Test P-Value : 0.1336144
##
##             Sensitivity : 1.0000
##             Specificity : 0.6923
##          Pos Pred Value : 0.8261
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5938
##          Detection Rate : 0.5938
##    Detection Prevalence : 0.7188
##       Balanced Accuracy : 0.8462
##
##        'Positive' Class : 0
##
```

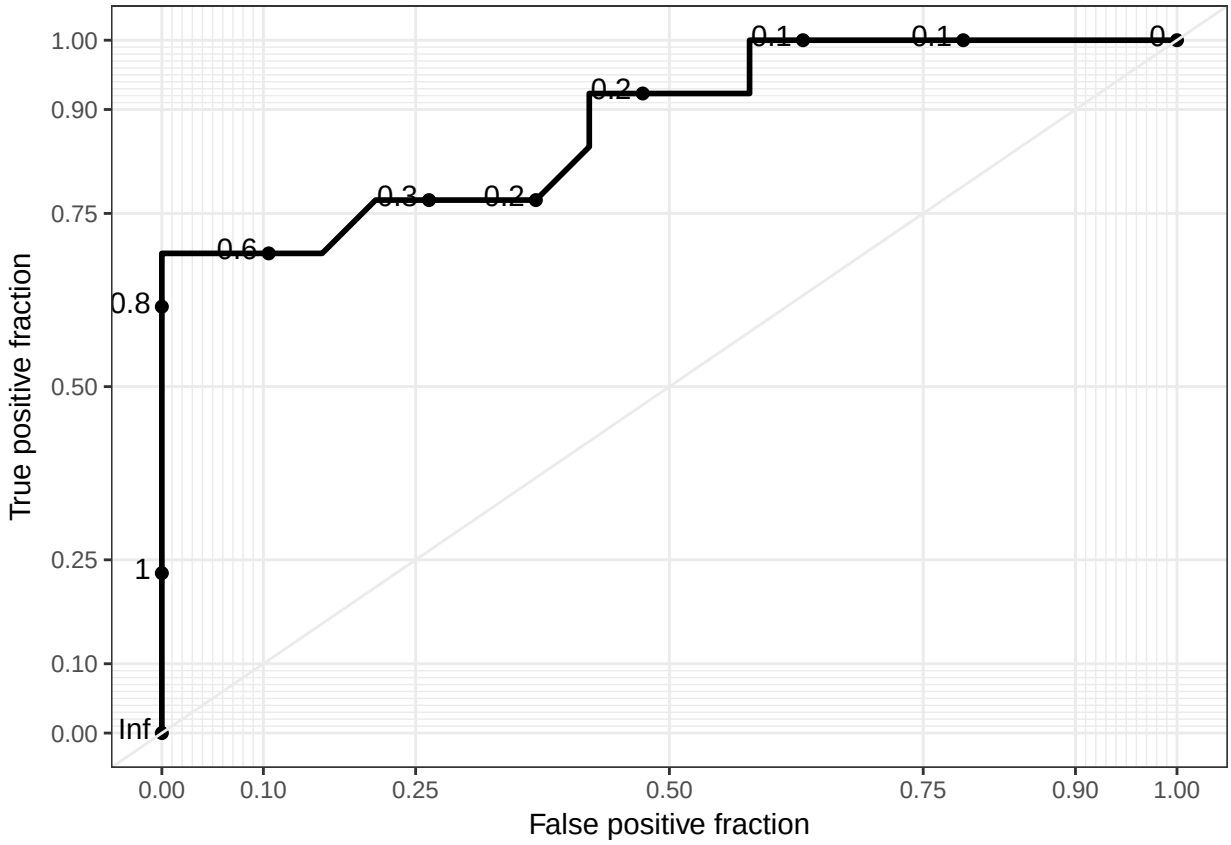
Receiver Operating Characteristics (ROC) Curve

Another way to evaluate model performance is to look at the ROC curve. It shows the sensitivity/specificity as a function of the cut off value.

```
# Calculate the ROC curve
mtcars_am_roc <- roc(am~prob, mtcars)
# You can call the calculation to see the area under curve
mtcars_am_roc
```

```
##
## Call:
## roc.formula(formula = am ~ prob, data = mtcars)
##
## Data: prob in 19 controls (am 0) < 13 cases (am 1).
## Area under the curve: 0.8785
```

```
# Or use the plot function to plot the curve
#plot(mtcars_am_roc)
# Ggplot with plotROC package does a better job.
ggplot(mtcars, aes(d=am, m=prob))+ geom_roc() + style_roc()
```



Looking at the curve, the ideal ROC curve would look like a right angle. Where Specificity is equal to 1 and sensitivity (1-false positive rate) is equal to 1. In which case the area under the curve would equal 1.

The worst case scenario is both sensitivity and specificity being equal to 0. In which case the area under the curve will be equal to 0.

Generally speaking, if the area under the curve is less than .6 then the model is not good.

Multinomial Logit

What if you have more than two categories? If you want to stick to regression models, what you need if you have multiple categories is to use Multinomial Logit. Multinomial logit is sometimes also referred to as softmax in more machine learning oriented circles.

Dataset

The dataset we will use comes from an online gaming platform:

```
# Read in csv file
games <- read.csv("data/games.csv")
# Inspect Structure of the variables
str(games)
```

```
## 'data.frame':    358 obs. of  7 variables:
##  $ appID    : int  280 1250 2300 3340 3960 4770 4870 4890 6200 7670 ...
##  $ dateMonth: int  201601 201601 201601 201601 201601 201601 201601 201601 201601 201601 ...
##  $ status   : Factor w/ 3 levels "Niche","Popular",..: 3 3 1 1 1 2 1 1 1 3 ...
##  $ avgMin   : num  75.5 2103.4 33.7 146.6 98 ...
##  $ listPrice: num  9.99 19.99 4.99 4.99 4.99 ...
##  $ reviews  : num  83 96 90 10 36.8 ...
##  $ multi    : int  0 1 1 0 1 1 0 1 0 0 ...
```

```
# Inspect descriptives
summary(games)
```

```
##      appID          dateMonth            status        avgMin
##  Min.   :   280   Min.   :201601   Niche    :323   Min.   :   0.00
##  1st Qu.:205796   1st Qu.:201601   Popular  : 26   1st Qu.:   0.00
##  Median :263040   Median :201601   SuperStar:  9   Median :  80.86
##  Mean   :231540   Mean   :201601                   Mean   : 274.98
##  3rd Qu.:315542   3rd Qu.:201601                   3rd Qu.: 223.34
##  Max.   :423880   Max.   :201601                   Max.   :9365.34
##
##    listPrice         reviews          multi
##  Min.   : 0.000   Min.   :10.00   Min.   :0.0000
##  1st Qu.: 4.987   1st Qu.:57.46   1st Qu.:0.0000
##  Median : 9.987   Median :75.57   Median :0.0000
##  Mean   :11.493   Mean   :68.77   Mean   :0.3212
##  3rd Qu.:14.987   3rd Qu.:89.00   3rd Qu.:1.0000
##  Max.   :89.990   Max.   :98.00   Max.   :1.0000
##  NA's   :11       NA's   :37
```

appID: Unique identifier for the game. dateM: Observation date (201601 in this set) status: Three level categorical variable indicating game's popularity. avgMin: Average number of minutes players played this game listPrice: List price. reviews: Positive reviews for the game. multi: Binary variable indicating multiplayer features.

Estimate The Model

R offers many alternatives in estimating multinomial logit models. I prefer mlogit package in my day to day use, while it is an overkill for simple multinomial logit models it also comes with nifty features (like reporting p values).

You will notice the overkill aspect of mlogit package in the function call. Don't worry too much about parameters that are not immediately apparent.

```
games_mlgt_0 <- mlogit(status ~ 0 | avgMin + multi, data = games, choice = "status", shape = "wide")
```

Let us break down the function call here:

```
games_mlgt_0 <- mlogit(status ~ 0 | avgMin + multi, data = games, choice = "status", shape = "wide")
```

1. games_mlgt_0 <- : store the estimation results in a variable called games_mglt_0
2. mlogit(status ~ 0 | avgMin + multi, data = games, choice = "status", shape = "wide")
   - status ~ 0 | avgMin + multi : formula saying estimate *status* based on *avgMin* and *multi*. The 0 | specifies that there are no alternative specific variables (don't worry).
   - data = games : where to find the variables.
   - choice = "status", shape = "wide": Telling mlogit package about data structure.

The function call is complicated due to the nature of mlogit. This package is designed for estimating a broad range of multinomial logit models (e.g. mixed multinomial logit) therefore the formula specification and data structure are overly complicated. I would not worry about these unless you are planning to use this model anytime soon.

Evaluate Results

Let's take a look at the results.

```
summary(games_mlgt_0)
```

```
##
## Call:
## mlogit(formula = status ~ 0 | avgMin + multi, data = games, choice = "status",
##     shape = "wide", method = "nr", print.level = 0)
##
## Frequencies of alternatives:
##     Niche   Popular SuperStar
##  0.902235  0.072626  0.025140
##
## nr method
## 6 iterations, 0h:0m:0s
## g'(-H)^-1g = 1.43E-07
## gradient close to zero
##
## Coefficients :
##                          Estimate  Std. Error  t-value  Pr(>|t|)
## Popular:(intercept)   -2.96410266  0.28646005 -10.3474 < 2.2e-16 ***
## SuperStar:(intercept) -4.33992506  0.51775019  -8.3823 < 2.2e-16 ***
## Popular:avgMin         0.00094383  0.00029144   3.2385  0.001202 **
## SuperStar:avgMin       0.00149670  0.00034211   4.3749 1.215e-05 ***
## Popular:multi          0.35618393  0.43915904   0.8111  0.417332
## SuperStar:multi       -0.48340590  0.88600051  -0.5456  0.585338
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log-Likelihood: -117.33
## McFadden R^2:  0.12808
## Likelihood ratio test : chisq = 34.471 (p.value = 5.9657e-07)
```

Evaluate Model Fit

What you learned about evaluating model fit thus far holds here. One nice thing about mlogit package is that it reports most statistics you will need to evaluate model fit automatically.

Here we see the LR test results are statistically significant (p value $< .05$). This means model is a significant improvement over null model.

We can not calculate regular R^2 for these types of models but there exists pseudo R^2 metrics. McFadden's R^2 shows how much of the variance is explained by the model.

Evaluate Coefficients

If you recall that our dependent variable had three levels (niche, popular, superstar) you will note that coefficients are reported only for two of them (popular, superstar). This is because the first level (niche) is being used as a baseline. Thus each coefficient is in comparison to the baseline.

Putting it in an equation:

$$ln\left(\frac{P(popular)}{P(niche)}\right) = \beta_{10} + \beta_{11} \times avgMin + \beta_{12} \times multi$$

$$ln\left(\frac{P(superstar)}{P(niche)}\right) = \beta_{20} + \beta_{21} \times avgMin + \beta_{22} \times multi$$

If you exponentiate the both sides, you will see that one minute increase in avgMin leads to 1.0009443 times increase in odds of being a popular game over being a niche game.

Here is a simple exercise, calculate the impact of 10 minute increase in avg min on odds of being a popular game over being a niche game.

Predicting Probabilities

Here is another example where using mlogit is complicating things. Mlogit package requires data to follow a distinct pattern. Hence we need to create one row of data for each option to be able to predict probabilities.

Here I will create a new dataset for three observations to be predicted. Since we have 3 levels in *status* variable, we will need 9 rows. I will vary the level of avgMin to investigate its effect on probabilities.

```
gtmp <- games[1,]
gtmp[1:9, ] <- games[1,]
# Half of the average minutes
gtmp[1:3, "avgMin"] <- mean(games$avgMin) / 2
# Average minutes
gtmp[4:6, "avgMin"] <- mean(games$avgMin)
# Twice the average minutes
gtmp[7:9, "avgMin"] <- mean(games$avgMin) * 2
# Let us look at the data we created
gtmp
```

```
##   appID dateMonth    status   avgMin listPrice reviews multi
## 1   280    201601 SuperStar 137.4875  9.986786      83     0
## 2   280    201601 SuperStar 137.4875  9.986786      83     0
## 3   280    201601 SuperStar 137.4875  9.986786      83     0
## 4   280    201601 SuperStar 274.9750  9.986786      83     0
## 5   280    201601 SuperStar 274.9750  9.986786      83     0
## 6   280    201601 SuperStar 274.9750  9.986786      83     0
## 7   280    201601 SuperStar 549.9500  9.986786      83     0
```

```
## 8   280    201601 SuperStar 549.9500  9.986786      83     0
## 9   280    201601 SuperStar 549.9500  9.986786      83     0
```

Now let us predict the probabilities.

```
predict(games_mlgt_0, gtmp)
```

```
##          Niche    Popular  SuperStar
## [1,] 0.9304284 0.05466956 0.01490203
## [2,] 0.9203234 0.06156859 0.01810801
## [3,] 0.8957233 0.07767921 0.02659745
```

You can see that the probability of being in popular/superstar categories increase as average minutes increases.

Solutions to Exercises

1 - Calculate the odds and log odds of critical hit.

```
p <- .05
1/(1-p) # Odds
```

```
## [1] 1.052632
```

```
log(1/(1-p)) # Log Odds
```

```
## [1] 0.05129329
```

2 - Calculate the probability event, if the log odds are .5?

```
lnOdds <- .5
exp(lnOdds) # Odds
```

```
## [1] 1.648721
```

```
exp(lnOdds) / (1 + exp(lnOdds)) # Probability
```

```
## [1] 0.6224593
```

3 - Calculate the impact of 10 minute increase in avg min on odds of being a popular game.

```
exp(coef(games_mlgt_0)[3]*10)
```

```
## Popular:avgMin
##       1.009483
```