Multiple regression, logistic regression

Dept of Security and Crime Science, UCL

B Kleinberg

Aims of this notebook

- multiple regression
  - recap
  - model selection
  - model comparison
- logistic regression
  - link function
  - fitting the GLM
  - interpreting the model
  - model comparison

Requirements

We assume that you have:

- read the required literature for weeks 1-3
- revised the lectures
- completed the introductory R tutorials (12 steps & How to solve problems) as well as the tutorial from week 2
- completed the homework in this module so far
- replicated the code from the lectures (if concepts/R implementation is unclear)

If you struggle with basics of R, you may also find this online book useful: https://bookdown.org/ndphillips/YaRrr/

Multiple regression

Load the `fraud_data` dataset using the `load` command. The dataset is located in the folder `./data` (assuming you are in the homework folder) and contains 1000 cases of employee fraud in financial trading companies.

The columns of this dataset are:

- damage: damage caused in USD
- gender: 0 = female, 1 = male
- promoted: whether the employee was promoted in the past 5 years (0 = no, 1 = yes)
- years_experience: number of years of overall job experience in this or a related position

Task: multiple regression recap

Build a regression model that models the damage cause through the employees' gender and promotion status.

```r
#your code here
```

What is the mean absolute error (MAE) of your model?

```r
#your code here
```

Now add the additional predictor variable `years_experience`. How does this affect your model's MAE? Did you expect this?

```r
#your code here
```

Finally, build the full model (i.e. include all main effects and interaction effects). Plot the residuals of that model against the observed values.

```r
#your code here
```

Task: multiple regression model selection

You can decide empirically, which combination of predictors results in the best model fit.

Start by building both a "null" model (i.e. only the intercept) and a "full" model (also called the saturated model).

```r
#your code here
```

Now start from the full model and use backwards stepwise regression. Which model does this procedure result in?

```r
#your code here
```

Do the same with forward stepwise regression. Does this result in the same model?

```r
#your code here
```

Finally, try to implement the bidirectional stepwise regression (hint: use "both" for the `direction` argument).

```r
#your code here
```

What do these findings tell you?

Task: multiple regression model comparison

You will often end up with different models of the same outcome variable. If these models are nested (i.e. one model can be derived from the other by removing model parameters), then you can use inferential statistics to determine whether one model is significantly worse than another.

In R, you can use the `anova` function to conduct an analysis of variance on two models to determine whether a simpler model is significantly worse than a more complex model.

Perform the model comparison test between (1) the full model vs the null model, (2) a 'gender-only' model and a 'gender + promotion' model, and (3) between the full model and the 'gender + promotion' model.

```r
#your code here
```

Rank the models from best to worst (use equal ranks if there is no significant difference).

Ranks come here:

1. ...
2. ...
3. ... ...

Logistic regression

Often you want to build a model to either understand relationships in the data, make predictions, or both, about an outcome variable that is scored as 0/1, present/not present, arrested/not arrested, etc. If such an outcome variable has only two levels, we also speak of a binary or dichotomous outcome.

Regression models can be applied in this context too. To understand what the special issue with binary outcome variables is, let's have a look at a dataset.

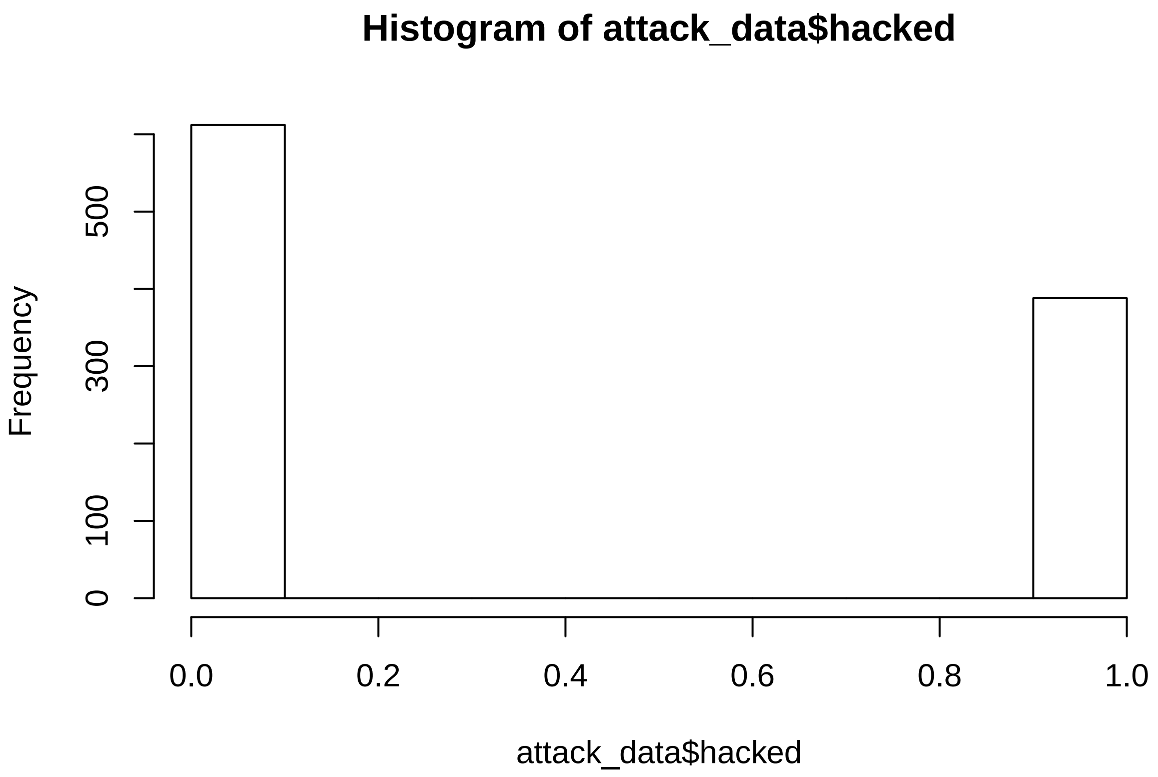
Load the `attack_data` dataset from `./data`. We use this dataset to revise concepts from the lecture. This dataset represents whether or not a website was hacked and the number of attempted hacking attacks Columns are:

- hacked: 0 = no, 1 = yes
- attempts

Task: logistic regression - link function

Suppose you want to model the relationship between `hacked` and `attempts`. If you look at the plot, you see that these data do not stem from a normal distribution:

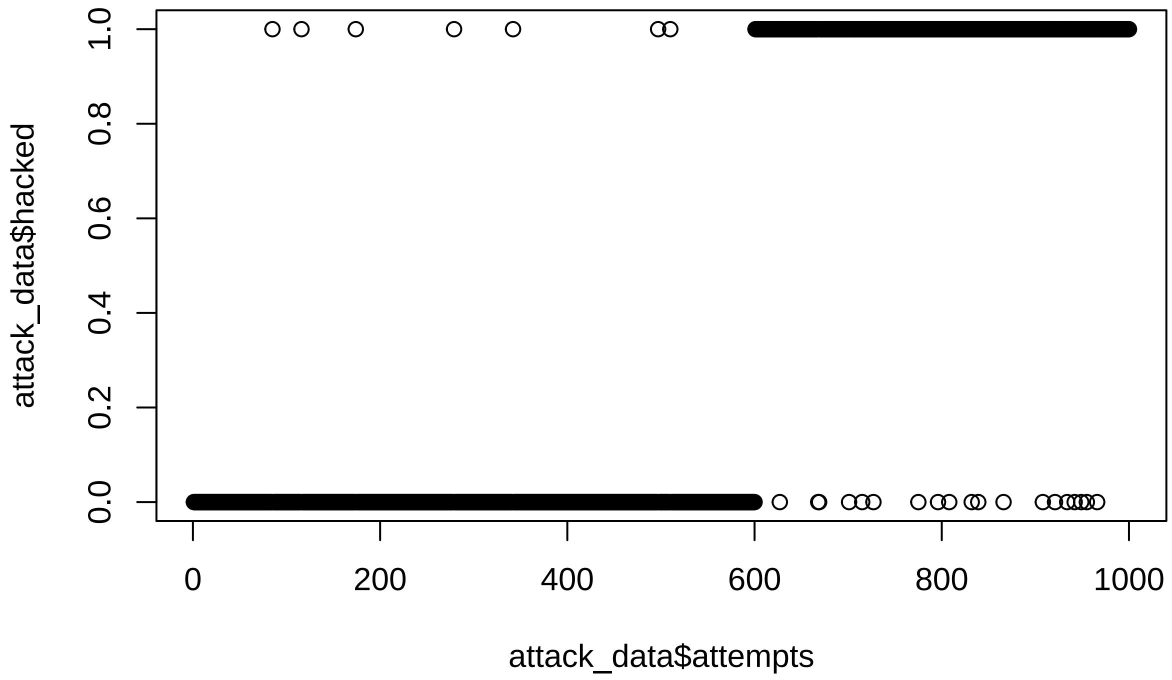
```r
load('./data/attack_data.RData')
hist(attack_data$hacked)
```



A distribution of a variable that can take only 0 and 1,

We can also look at the 'raw' data to get a better understanding of the relationship between the variables `hacked` and `attempts`:

```r
plot(attack_data$attempts, attack_data$hacked)
```



So let's start with what we know from regression modelling and 'fit' an ordinary linear model:

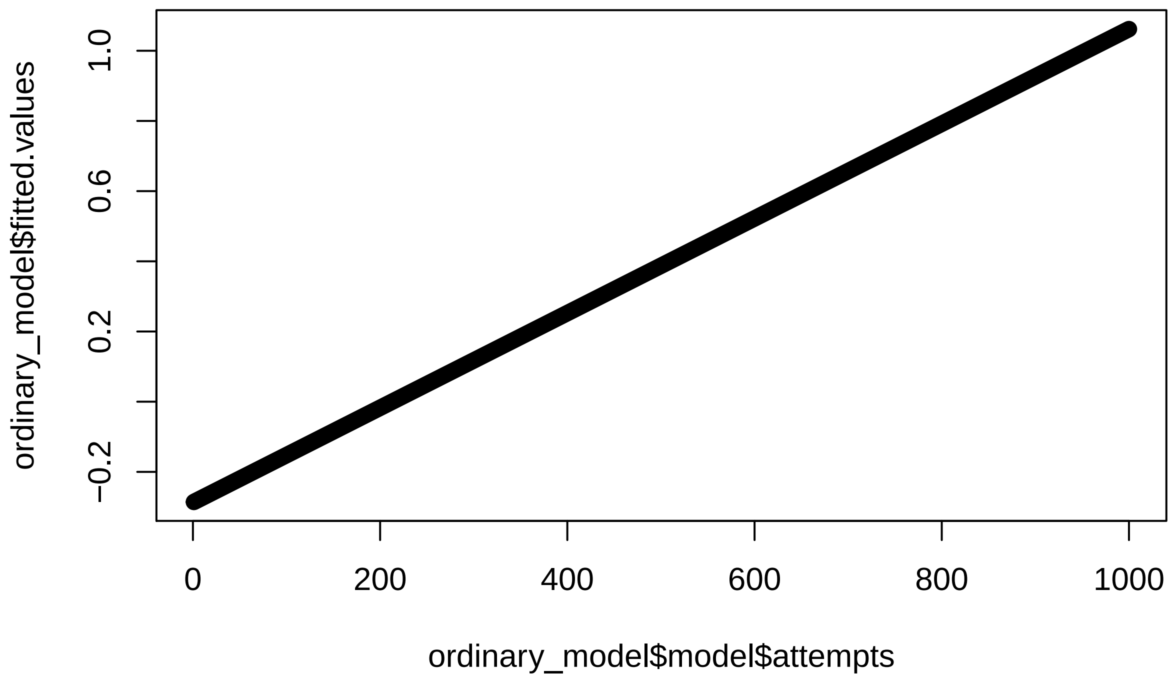
```
ordinary_model = glm(hacked ~ attempts
                     , data = attack_data
                     , family = gaussian
                    )
summary(ordinary_model)
```

```
##
## Call:
## glm(formula = hacked ~ attempts, family = gaussian, data = attack_data)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.01593  -0.20825   0.03781   0.21227   1.17248
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.871e-01  1.856e-02  -15.47   <2e-16 ***
## attempts     1.349e-03  3.212e-05   41.99   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.08599304)
##
##     Null deviance: 237.456  on 999  degrees of freedom
## Residual deviance:  85.821  on 998  degrees of freedom
## AIC: 388.39
##
## Number of Fisher Scoring iterations: 2
```

Note that a GLM with family "gaussian" is identical to a normal linear model. This is because the ordinary linear model assumes that the outcome variable is normally distributed (i.e. follows a Gaussian distribution).

To see what might be problematic, have a look at the predicted values:

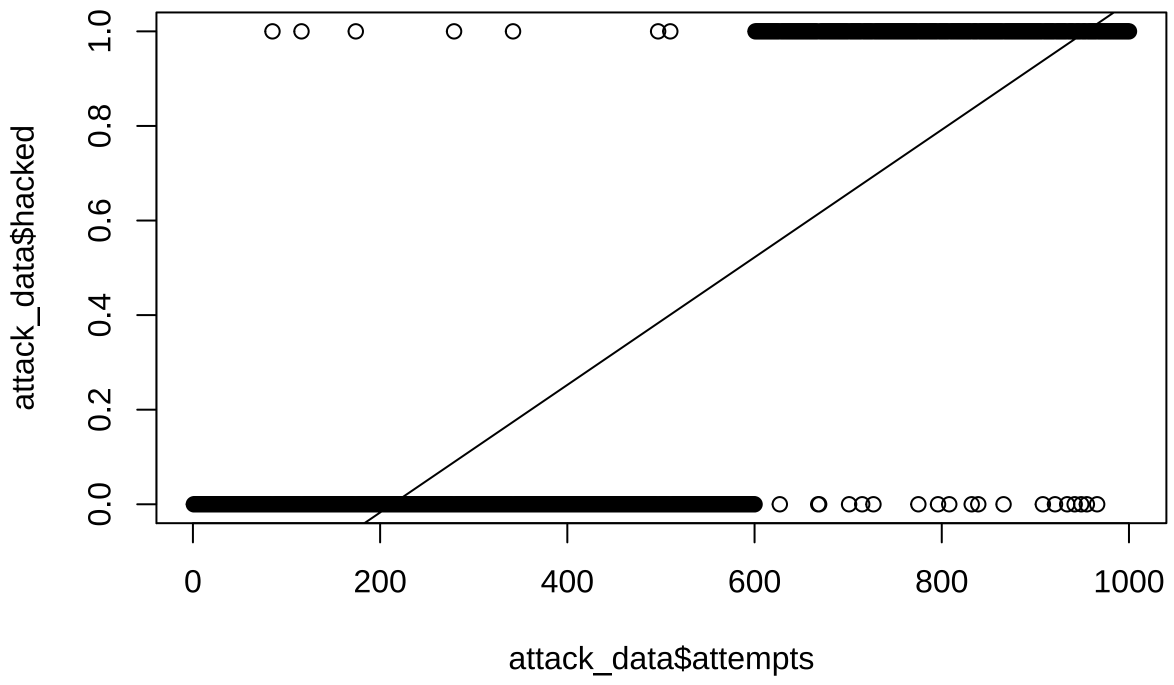
```r
plot(ordinary_model$model$attempts, ordinary_model$fitted.values)
```



You will notice that the values predicted through the model are (1) not only 1s and 0s and (2) exceed 1 and are even smaller than 0. Clearly, for a dataset where the outcome variable can only take the value 0 and 1, this is an inadequate way to model the data.

You can also look at the actual regression line fitted to the raw data:

```r
{plot(attack_data$attempts, attack_data$hacked)
  abline(ordinary_model)}
```



So we need a solution to that issue.

Luckily, there is a way to transform the 1/0 outcome variable to a continuous variable so that the model can make predictions on a continuous scale.

A neat way to do this, is the logit function that performs the following steps:

1. it assumes that each case has a probability of being 1 or 0

Let's do this for a sequence fro 0.0 to 1.0 in steps of 0.1

```r
probabilities = seq(from = 0.01, to = 0.99, by=0.01)
```

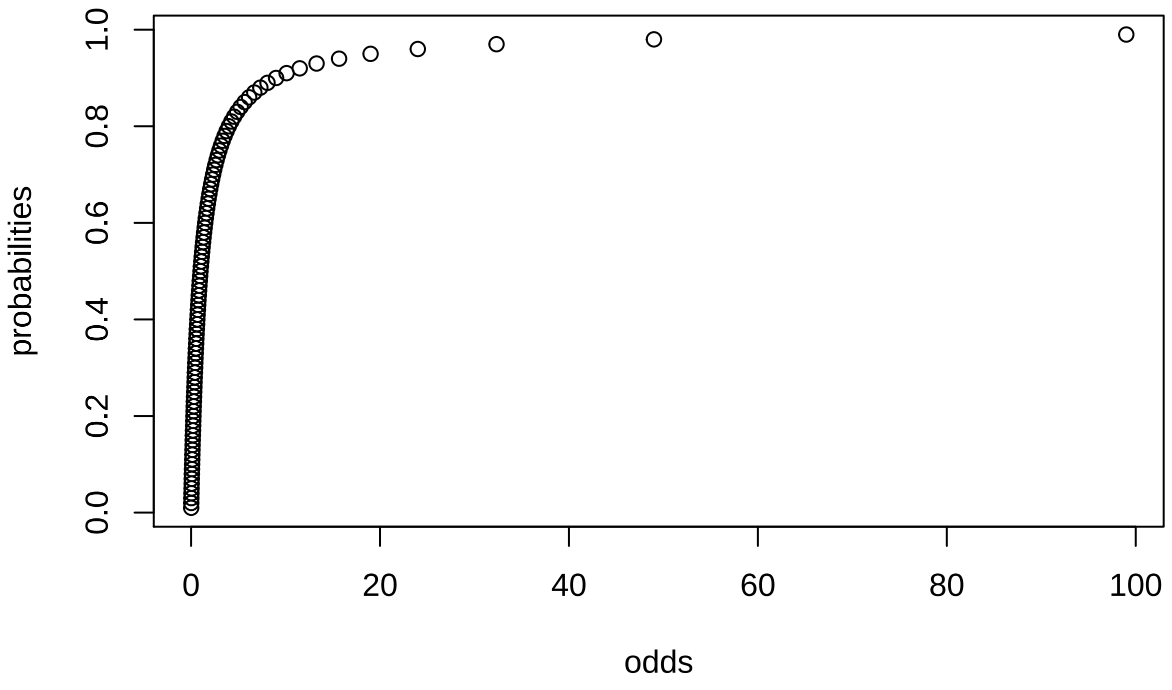
This brings the transformed outcome variable from a 0,1 scale to a 0:1 scale (note: the `:` reads as "to").

2. it transforms these probabilities to the odds: `odds = P/(1-P)`

```r
odds = probabilities/(1-probabilities)
```

This transforms the outcome variable to a scale ranging from 0:Inf.

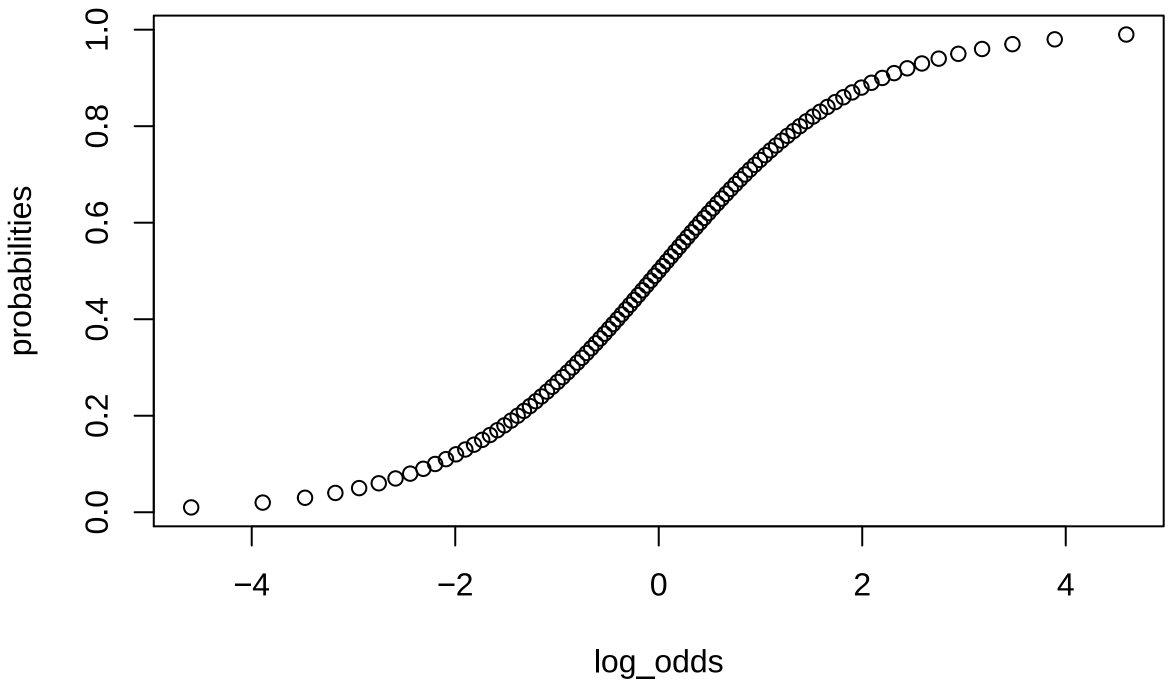
```r
plot(odds, probabilities)
```



3. it calculates the logarithm of the odds (if you need to recap the concept of a logarithm, plese take a look at this video tutorial). More specifically, we take the natural logarithm.

```r
log_odds = log(odds)
```

```r
plot(log_odds, probabilities)
```



You can see that now we transformed the outcome variable further to a scale ranging from -Inf:Inf.

It is this logit function that is used to transform the outcome variable from a 1,0 discrete range to a continuous range from -Inf:Inf.

You will see this in action further below…

In the GLM function, you can specify this by using the `family =` argument and setting it to "binomial" (since our outcome variable stems from a binomial distribution).

Task: logistic regression - fitting the GLM

Build a logistic regression model that models whether or not a website was hacked through the number of attacks:

```
#your code here
```

Task: logistic regression - interpreting the model

Take a look at the model summary. Remember what the logit model does? If we model the "log-odds", then the coefficients (what we call the intercept and slope in linear regression) need to be interpreted as such.

But because the log-odds are hard to interpret, we want to transform them back to the more interpretable odds.

From the video above, you will have learned that you can reverse the natural logarithm by taking e to the power of the logarithm.

Do this transformation:

```
#your code here
```

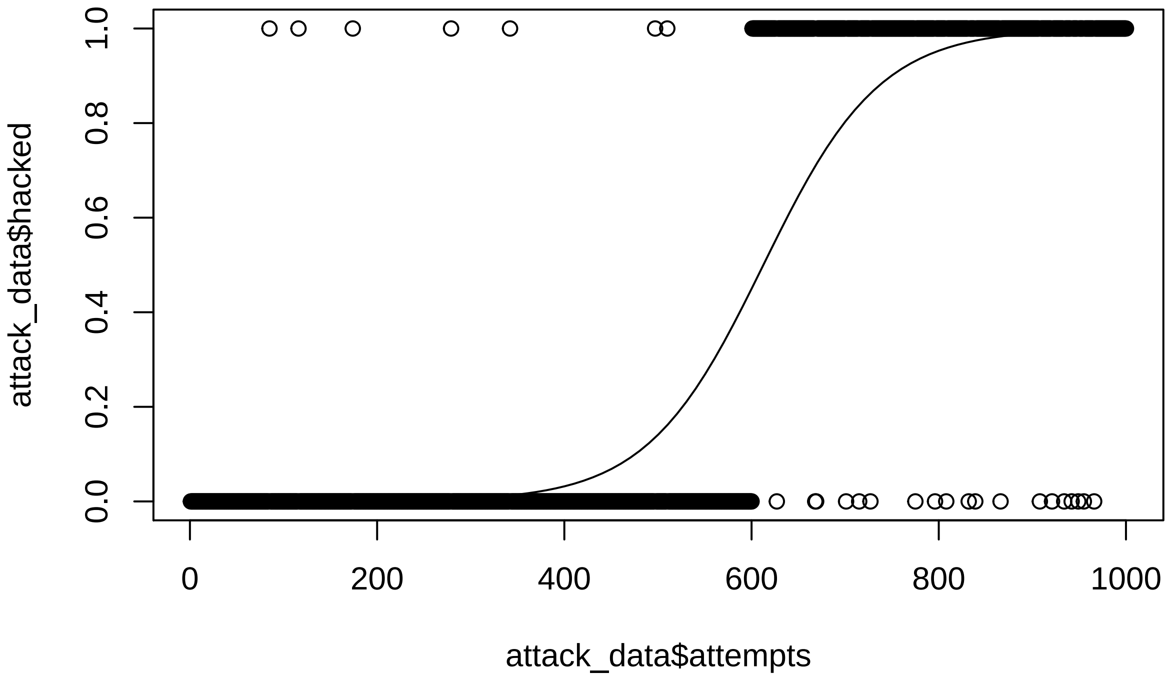
What does this yield (i.e. how do you intepret these values)?

What would the interpretation of these findings look like in your own words?

Task: logistic regression - curve fitting

Similar to the "line-fitting" of linear regression, we can also look at the fitted model visually.

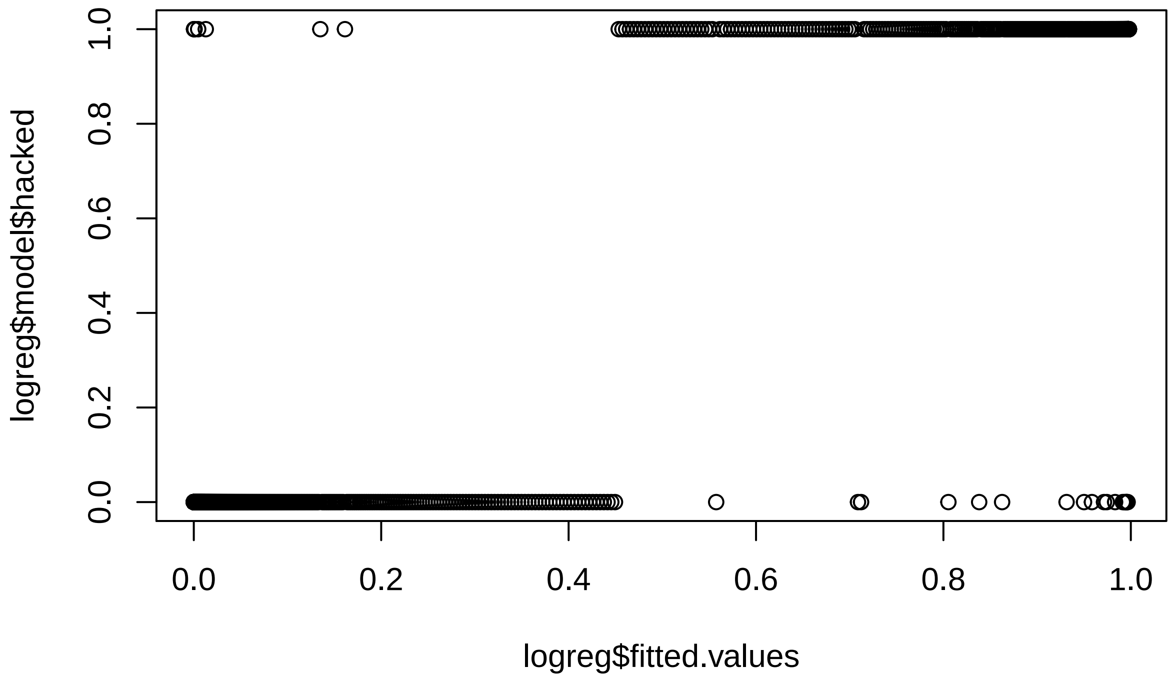
```
logreg = glm(hacked ~ attempts
             , family=binomial
             , data = attack_data)
{plot(attack_data$attempts, attack_data$hacked)
  curve(predict(logreg
                , data.frame(attempts=x)
                , type="resp")
        , add=TRUE)}
```



You can see that the model (= the curve) predicts values exclusively in the 0:1 range. However, you can also see that while the majority of the predicted values is either 0 or 1, some values are in between (e.g. around attempts == 500). This is the reason why you need thresholds if you want to ascertain the accuracy of such a model. In Year 3, you will learn about applications of this thresholding for logistic regression in machine learning.

You can see the relationship between fitted values (i.e. probabilities of a case being in on eof the two outcome classes - hacked vs not hacked - given a certain number of attempts) and the observed values:

```
plot(logreg$fitted.values, logreg$model$hacked)
```



END