

# Binomial Generalized Linear Models

Filippo Gambarota

University of Padova

2022/2023

Updated on 2023-05-02

# Outline

1. Binomial GLM
2. Binomial GLM - Parameter Intepretation
3. Binomial GLM - Inference
4. Binomial GLM - Plotting effects
5. GLM - Diagnostic
6. Binomial GLM - Probit link

## Binomial GLM

# Example: Passing the exam

We want to measure the impact of **watching tv-shows** on the probability of **passing the statistics exam**.

- **exam**: **passing the exam** (1 = “passed”, 0 = “failed”)
- **tv\_shows**: **watching tv-shows regularly** (1 = “yes”, 0 = “no”)

```
head(dat)
```

```
##   tv_shows exam
## 1         1    0
## 2         1    1
## 3         1    1
## 4         1    0
## 5         1    1
## 6         1    0
```

# Example: Passing the exam

We can create the **contingency table**

```
xtabs(~exam + tv_shows, data = dat) |>  
  addmargins()
```

```
##      tv_shows  
## exam    0    1 Sum  
##    0    37   21  58  
##    1    13   29  42  
##   Sum    50   50 100
```

# Example: Passing the exam

Each cell probability  $\pi_{ij}$  is computed as  $\pi_{ij}/n$

```
(xtabs(~exam + tv_shows, data = dat)/n) |>  
  addmargins()
```

```
##      tv_shows  
## exam      0      1  Sum  
##   0  0.37 0.21 0.58  
##   1  0.13 0.29 0.42  
##   Sum 0.50 0.50 1.00
```

## Example: Passing the exam - Odds

The most common way to analyze a 2x2 contingency table is using the **odds ratio** (OR). Firstly let's define *the odds of success* as:

$$\begin{aligned} odds &= \frac{\pi}{1 - \pi} \\ \pi &= \frac{odds}{odds + 1} \end{aligned}$$

- the **odds** are non-negative, ranging between 0 and  $+\infty$
- an **odds** of e.g. 3 means that we expect 3 *success* for each *failure*

# Example: Passing the exam - Odds

For the exam example:

```
odds <- function(p) p / (1 - p)
p11 <- mean(with(dat, exam[tv_shows == 1])) # passing exam / tv_shows
odds(p11)
```

```
## [1] 1.380952
```



# Example: Passing the exam - Odds Ratio

The OR is a ratio of odds:

$$OR = \frac{\frac{\pi_1}{1-\pi_1}}{\frac{\pi_2}{1-\pi_2}}$$

- OR ranges between 0 and  $+\infty$ . When  $OR = 1$  the odds for the two conditions are equal
- An e.g.  $OR = 3$  means that being in the condition at the numerator increase 3 times the odds of success

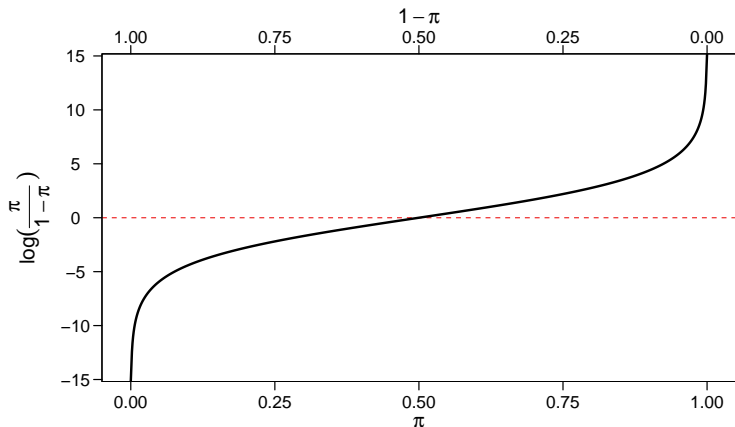
# Example: Passing the exam - Odds Ratio

```
odds_ratio <- function(p1, p2) odds(p1) / odds(p2)
p11 <- mean(with(dat, exam[tv_shows == 1])) # passing exam / tv_shows
p10 <- mean(with(dat, exam[tv_shows == 0])) # passing exam / not tv_shows
odds_ratio(p11, p10)
```

```
## [1] 3.930403
```

# Why using these measure?

The odds have an interesting property when taking the logarithm. We can express a probability  $\pi$  using a scale ranging  $[-\infty, +\infty]$



# Another example, **Teddy Child**

We considered a Study conducted by the University of Padua (TEDDY Child Study, 2020)<sup>1</sup>. Within the study, researchers asked the participants (mothers of a young child) about the presence of post-partum depression and measured the parental stress using the PSI-Parenting Stress Index.

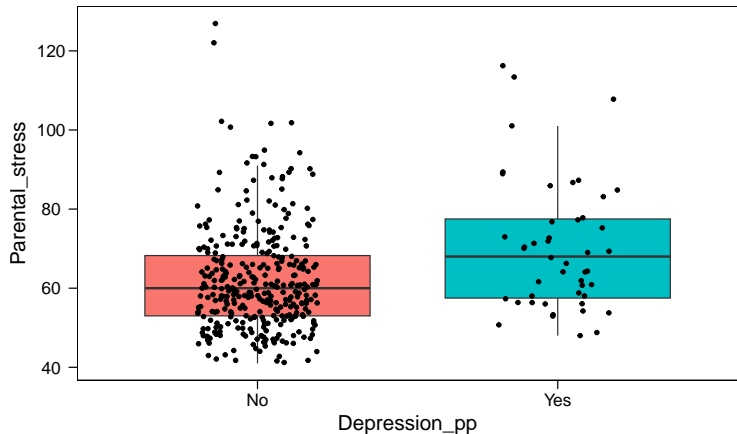
ID	Parental.stress	Depression.pp
1	75	No
2	51	No
3	76	No
4	88	No
...	...	...
376	67	No
377	71	No
378	63	No
379	70	No

---

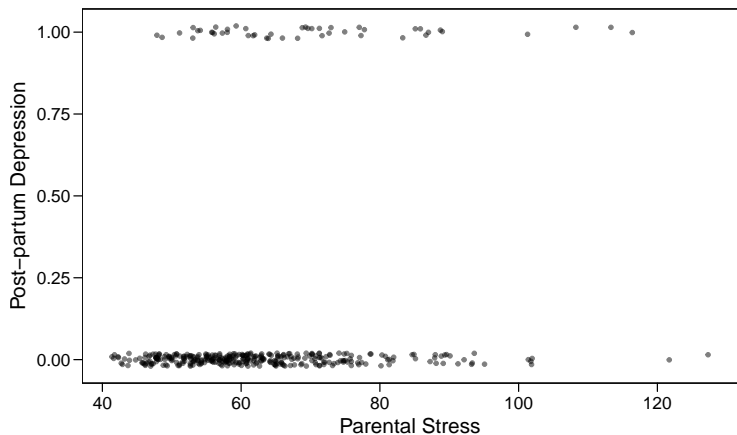
<sup>1</sup>Thanks to Prof. Paolo Girardi for the example, see <https://teddychild.dpss.psy.unipd.it/> for information

# Another example, **Teddy Child**

We want to see if the parental stress increase the probability of having post-partum depression:



## Another example, **Teddy Child**



# Another example, Teddy Child

Let's start by fitting a linear model `Depression_pp ~ Parental_stress`. We consider “Yes” as 1 and “No” as 0.

```
fit_lm <- lm(Depression_pp01 ~ Parental_stress, data = teddy)
summary(fit_lm)
```

```
##
## Call:
## lm(formula = Depression_pp01 ~ Parental_stress, data = teddy)
##
## Residuals:
```

	Min	1Q	Median	3Q	Max
##	-0.42473	-0.13768	-0.10003	-0.05768	0.94702

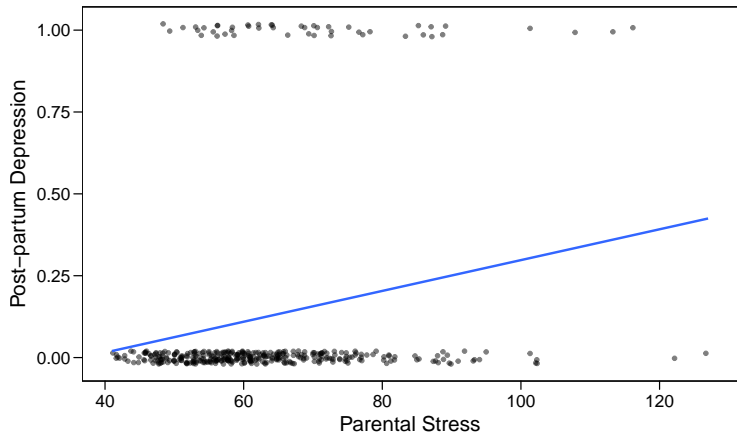
```
##
## Coefficients:
```

	Estimate	Std. Error	t value	Pr(> t )
## (Intercept)	-0.172900	0.077561	-2.229	0.026389 *
## Parental_stress	0.004706	0.001201	3.919	0.000105 ***

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3239 on 377 degrees of freedom
## Multiple R-squared:  0.03915,    Adjusted R-squared:  0.0366
## F-statistic: 15.36 on 1 and 377 DF,  p-value: 0.0001054
```

# Another example, **Teddy Child**

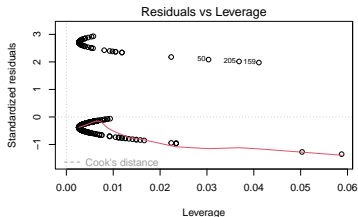
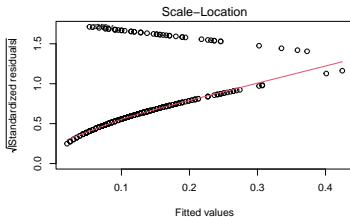
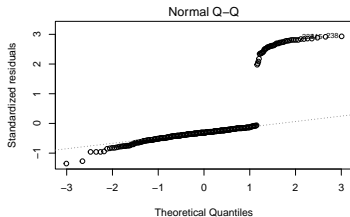
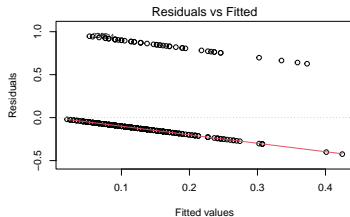
Let's add the fitted line to our plot:





# Another example, Teddy Child

... and check the residuals, pretty bad right?



# Another example, Teddy Child

As for the exam example, we could compute a sort of contingency table despite the `Parental_stress` is a numerical variable by creating some discrete categories (just for exploratory analysis):

```
table(teddy$Depression_pp, teddy$Parental_stress_c) |>  
  round(2)
```

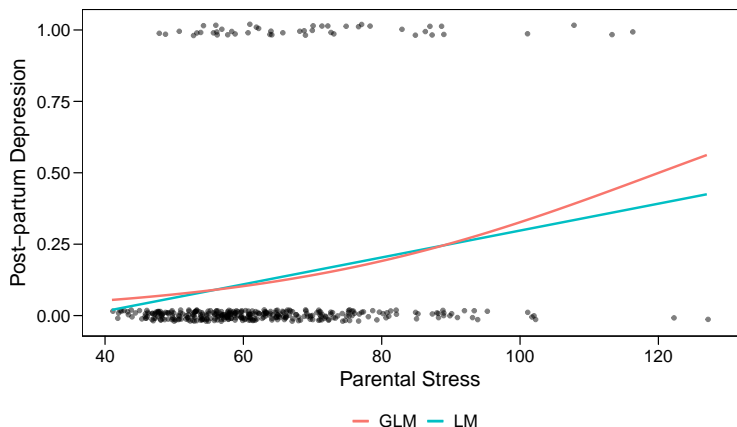
```
##  
##      < 40 40-60 60-80 80-100 > 100  
## No      0   164   136     26     6  
## Yes     0    15    21      7     4
```

```
table(teddy$Depression_pp, teddy$Parental_stress_c) |>  
  prop.table(margin = 2) |>  
  round(2)
```

```
##  
##      < 40 40-60 60-80 80-100 > 100  
## No      0.92 0.87  0.79 0.60  
## Yes     0.08 0.13 0.21 0.40
```

# Another example, **Teddy Child**

Ideally, we could compute the increase in the odds of having the post-partum depression as the parental stress increase. In fact, as we are going to see, the Binomial GLM is able to estimate the non-linear increase in the probability.



# Binomial GLM

- The **random component** of a Binomial GLM the binomial distribution with parameter  $\pi$
- The **systematic component** is a linear combination of predictors and coefficients  $\beta X$
- The **link function** is a function that map probabilities into the  $[-\infty, +\infty]$  range.

# Binomial GLM - Logit Link

The **logit** link is the most common link function when using a binomial GLM:

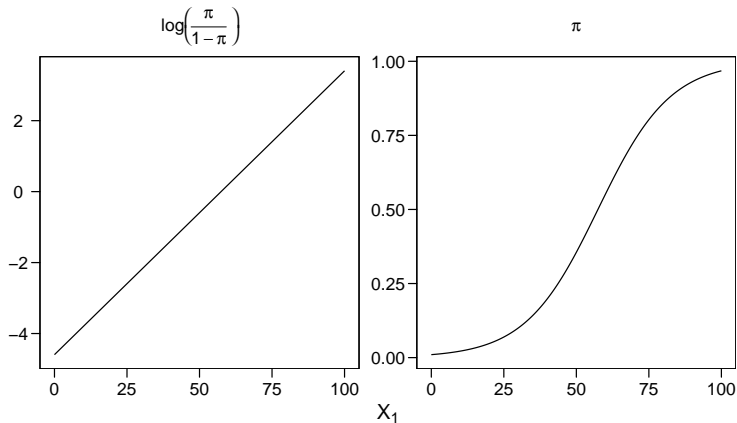
$$\log\left(\frac{\pi}{1-\pi}\right) = \beta_0 + \beta_1 X_1 + \dots \beta_p X_p$$

The inverse of the **logit** maps again the probability into the  $[0, 1]$  range:

$$\pi = \frac{e^{\beta_0 + \beta_1 X_1 + \dots \beta_p X_p}}{1 + e^{\beta_0 + \beta_1 X_1 + \dots \beta_p X_p}}$$

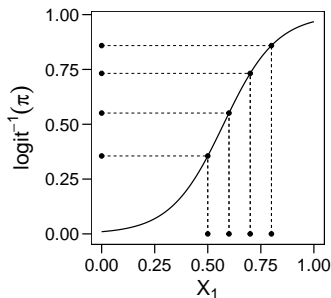
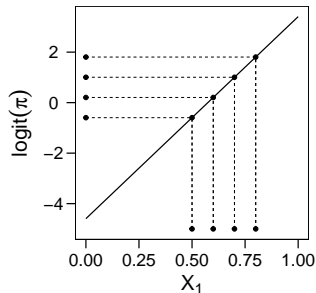
# Binomial GLM - Logit Link

Thus with a single numerical predictor  $x$  the relationship between  $x$  and  $\pi$  is non-linear on the probability scale but linear on the logit scale.

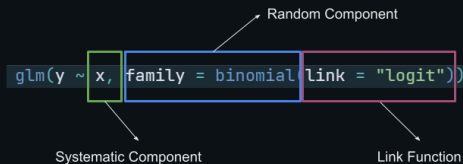


# Binomial GLM - Logit Link

The problem is that effects are non-linear, thus is more difficult to interpret and report model results



# Binomial GLM - Model fitting in R



The diagram shows the R function `glm(y ~ x, family = binomial, link = "logit")` with three colored boxes highlighting different parts: a green box around `x`, a blue box around `family = binomial`, and a pink box around `link = "logit"`. Arrows point from these boxes to labels: "Systematic Component" points to the green box, "Random Component" points to the blue box, and "Link Function" points to the pink box.

```
glm(y ~ x, family = binomial, link = "logit")
```

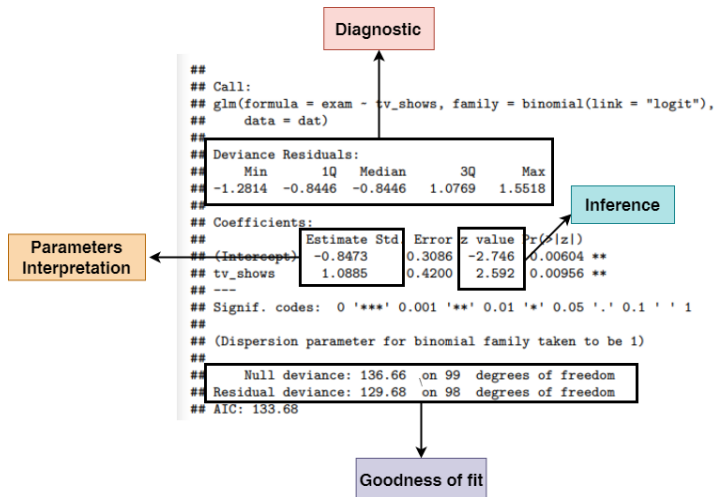
Random Component

Systematic Component

Link Function



# The big picture...



# Binomial GLM - Model fitting in R

We can model the contingency table presented before. We put data in **binary form**:

```
##      tv_shows
## exam 0  1
##      0 35 22
##      1 15 28
```

```
##      tv_shows exam
## 1          1    0
## 2          1    1
## 3          1    1
## 4          1    1
## 5          ...   ...
## 6          0    0
## 7          0    1
## 8          0    0
## 9          0    0
```

# Binomial GLM - Intercept only model

Let's start from the simplest model (often called null model) where there are no predictors:

```
fit0 <- glm(exam ~ 1, data = dat, family = binomial(link = "logit"))
summary(fit0)
```

```
##
## Call:
## glm(formula = exam ~ 1, family = binomial(link = "logit"), data = dat)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.060  -1.060  -1.060   1.299   1.299
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.2819     0.2020  -1.395   0.163
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 136.66  on 99  degrees of freedom
## Residual deviance: 136.66  on 99  degrees of freedom
## AIC: 138.66
##
## Number of Fisher Scoring iterations: 4
```

# Binomial GLM - Intercept only model

When fitting an intercept-only model, the parameter is the average value of the  $y$  variable:

$$\log\left(\frac{\pi}{1-\pi}\right) = \beta_0$$
$$\pi = \frac{e^{\beta_0}}{1 + e^{\beta_0}}$$

# Binomial GLM - Intercept only model

In R, the  $\text{logit}(\pi)$  is computed using `qlogis()` that is the `q + logis` combination of functions to work with probability distributions. The  $\text{logit}^{-1}$  thus the inverse of the logit function is `plogis()`:

```
# average y on the response scale  
mean(dat$exam)
```

```
## [1] 0.43  
c("logit" = coef(fit0)[1],  
  "inv-logit" = plogis(coef(fit0)[1])  
)
```

```
##      logit.(Intercept) inv-logit.(Intercept)  
##           -0.2818512           0.4300000
```

# Binomial GLM - Link function (TIPS)

If you are not sure about how to transform using the link function you can directly access the `family()` object in R that contains the appropriate link function and the corresponding inverse.

```
bin <- binomial(link = "logit")  
bin$linkfun() # the same as plogis  
bin$linkinv() # the same as qlogis
```

# Binomial GLM - Model with X

Now we can add the `tv_shows` predictor. Now the model has two coefficients. Given that the `tv_shows` is a binary variable, the intercept is the average  $y$  when `tv_shows` is 0, and the `tv_shows` coefficient is the increase in  $y$  for a unit increase in `tv_shows`:

```
fit <- glm(exam ~ tv_shows, data = dat, family = binomial(link = "logit"))
summary(fit)
```

```
##
## Call:
## glm(formula = exam ~ tv_shows, family = binomial(link = "logit"),
##      data = dat)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.2814  -0.8446  -0.8446   1.0769   1.5518
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.8473     0.3086  -2.746  0.00604 **
## tv_shows       1.0885     0.4200   2.592  0.00956 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 136.66  on 99  degrees of freedom
## Residual deviance: 129.68  on 98  degrees of freedom
## AIC: 133.68
##
```

# Binomial GLM - Model with X

Thinking about our data, the (Intercept) is the probability of passing the exam without watching tv-shows. The `tv_shows` (should be) the difference in the probability of passing the exam between people who watched or did not watched tv-shows, BUT:

- we are on the logit scale. Thus we are modelling **log(odds)** and not probabilities
- a difference on the **log** scale is a ratio on the raw scale. Thus taking the exponential of `tv_shows` we obtain the ratio of odds of passing the exam watching vs non-watching tv-shows. Do you remember something?



# Binomial GLM - Model with $X_1$

The `tv_shows` is exactly the Odds Ratio that we calculated on the contingency table:

```
# from model coefficients  
exp(coef(fit)["tv_shows"])
```

```
## tv_shows  
## 2.969697
```

```
# from the contingency table  
odds_ratio(mean(dat$exam[dat$tv_shows == 1]),  
           mean(dat$exam[dat$tv_shows == 0]))
```

```
## [1] 2.969697
```

## Binomial GLM - Parameter Interpretation

# Binomial GLM - Model Interpretation

Given the non-linearity and the link function, parameter interpretation is not easy for GLMs. In the case of the Binomial GLM we will see:

- interpreting model coefficients on the linear and logit scale
- odds ratio (already introduced)
- the divide by 4 rule [1], [2]
- marginal effects
- predicted probabilities

# Binomial GLM - Interpreting model coefficients

Models coefficients are interpreted in the same way as standard regression models. The big difference concerns:

- numerical predictors
- categorical predictors

Using contrast coding and centering/standardizing we can make model coefficients more interpretable or tailored to our research question.

# Binomial GLM - Categorical predictors

We use a categorical predictor with  $p$  levels, the model will estimate  $p - 1$  parameters. The interpretation of these parameters is controlled by the contrast coding. In R the default is the treatment coding (or dummy coding). Essentially R create  $p - 1$  dummy variables (0 and 1) where 0 is the reference level (usually the first category) and 1 is the current level. We can see the coding scheme using the `model.matrix()` function that return the  $X$  matrix:

```
##   X.Intercept. tv_shows
## 1           1         1
## 2           1         1
## 3           1         1
## 4           1         1
## 5           ...         ...
## 6           1         0
## 7           1         0
## 8           1         0
## 9           1         0
```

# Binomial GLM - Categorical predictors

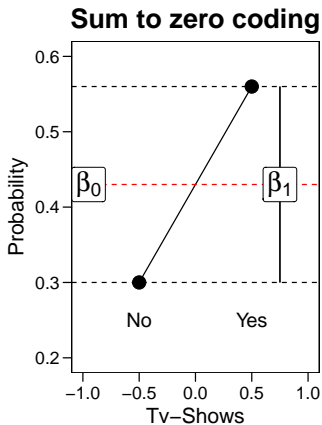
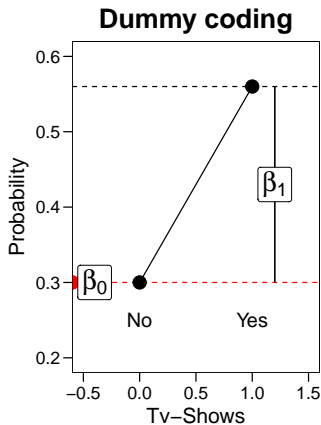
In the simple case of the `exam` dataset, the intercept ( $\beta_0$ ) is the reference level (default to 0 because is the first) and  $\beta_0$  is the difference between the actual level and the reference level. If we change the order of the levels we could change the intercept value while  $\beta_1$  will be the same. As an example we could use the so-called sum to zero coding where instead of assigning 0 and 1 we use different values. For example assigning -0.5 and 0.5 will make the intercept the grand-mean:

```
dat$tv_shows0 <- ifelse(dat$tv_shows == 0, -0.5, 0.5)
fit <- glm(exam ~ tv_shows0, data = dat, family = binomial(link = "logit"))
# grand mean
mean(c(mean(dat$exam[dat$tv_shows == 1]), mean(dat$exam[dat$tv_shows == 0])))
```

```
## [1] 0.43
# intercept
plogis(coef(fit)[1])
```

```
## (Intercept)
## 0.4248077
```

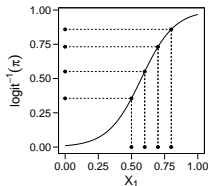
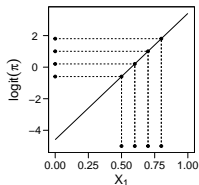
# Binomial GLM - Categorical predictors



# Binomial GLM - Numerical predictors

With numerical predictors the idea is the same as categorical predictors. In fact, categorical predictors are converted into numbers (e.e., 0 and 1 or -0.5 and 0.5). The only caveat is that the effects are linear only the **logit** scale. Thus  $\beta_1$  is interpreted in the same way as standard linear models only on the link-function scale. For the **binomial**

GLM the  $\beta_1$  is the increase in the  $\log(odds(\pi))$  for a unit-increase in the  $x$ . In the response (probability) scale, the  $\beta_1$  is the multiplicative increase in the odds of  $y = 1$  for a unit increase in the predictor.





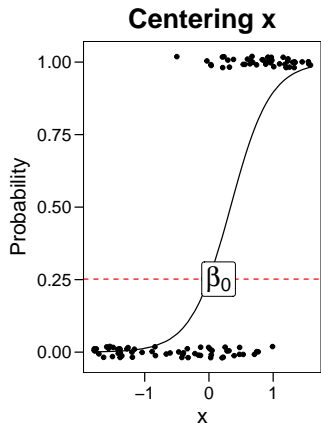
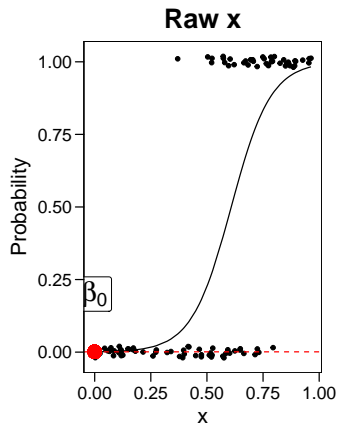
# Binomial GLM - Numerical predictors

With numerical predictors we could mean-center and or standardize the predictors. With centering (similarly to the categorical example) we change the interpretation of the intercept. Standardizing is helpful to have more meaningful  $\beta$  values. The  $\beta_i$  of a centered predictor is the increase in  $y$  for a increase in one standard deviation of  $x$ .

$$x_{cen} = x_i - \hat{x}$$

$$x_z = \frac{x_i - \hat{x}}{\sigma_x}$$

# Binomial GLM - Numerical predictors



# Binomial GLM - Numerical predictors

Let's return to our teddy child example and fitting the proper model:

```
fit_glm <- glm(Depression_pp01 ~ Parental_stress, data = teddy, family = binomial(link = "logit"))
summary(fit_glm)
```

```
##
## Call:
## glm(formula = Depression_pp01 ~ Parental_stress, family = binomial(link = "logit"),
##      data = teddy)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.2852  -0.5165  -0.4509  -0.3861   2.3096
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -4.323906   0.690689  -6.260 3.84e-10 ***
## Parental_stress  0.036015   0.009838   3.661 0.000251 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 284.13  on 378  degrees of freedom
## Residual deviance: 271.23  on 377  degrees of freedom
## AIC: 275.23
##
## Number of Fisher Scoring iterations: 5
```

# Binomial GLM - Numerical predictors

The (Intercept) ( $\beta_0$ ) is the probability of having post-partum depression for a mother with parental stress zero (maybe better centering?)

$$p(yes|x = 0) = g^{-1}(\beta_0)$$

```
plogis(coef(fit_glm)["(Intercept)"])
```

```
## (Intercept)  
## 0.01307482
```

# Binomial GLM - Numerical predictors

The Parental\_stress ( $\beta_1$ ) is the increase in the  $\log(odds)$  of having the post-partum depression for a unit increase in the parental stress index. If we take the exponential of  $\beta_1$  we obtain the increase in the *odds* of having post-partum depression for a unit increase in parental stress index.

```
exp(coef(fit_glm)["Parental_stress"])
```

```
## Parental_stress  
##      1.036671
```

# Binomial GLM - Numerical predictors

The problem is that, as shown before, the effects are non-linear on the probability scale while are linear on the logit scale. On the logit scale, all differences are constant:

```
pr <- list(c(10, 11), c(50, 51), c(70, 71))

predictions <- lapply(pr, function(x) {
  predict(fit_glm, newdata = data.frame(Parental_stress = x))
})

predictions

## [[1]]
##      1      2
## -3.963759 -3.927744
##
## [[2]]
##      1      2
## -2.523171 -2.487156
##
## [[3]]
##      1      2
## -1.802877 -1.766862
# notice that the difference is exactly the Parental_stress parameter
sapply(predictions, diff)

##      2      2      2
## 0.0360147 0.0360147 0.0360147
```

# Binomial GLM - Numerical predictors

While on the probability scale, the differences are not the same. This is problematic when interpreting the results of a Binomial GLM with a numerical predictor.

```
(predictions <- lapply(predictions, plogis))
```

```
## [[1]]
##      1      2
## 0.01863764 0.01930790
##
## [[2]]
##      1      2
## 0.07424969 0.07676350
##
## [[3]]
##      1      2
## 0.1415012 0.1459330
```

```
sapply(predictions, diff)
```

```
##      2      2      2
## 0.0006702661 0.0025138036 0.0044317558
```

# Binomial GLM - Divide by 4 rule

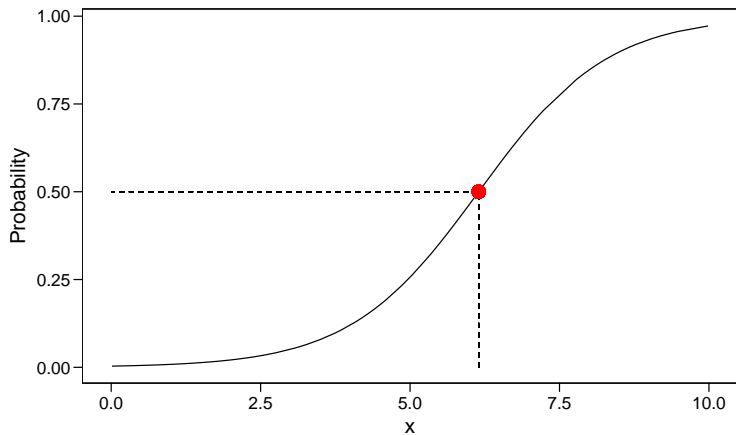
The **divide by 4 rule** is a very easy way to evaluate the effect of a continuous predictor on the probability.

Given the non-linearity, the derivative of the logistic function (i.e., the slope) is maximal when predicts probabilities around  $\sim 0.5$ .

In fact,  $\beta_i \pi(1 - \pi)$  is maximized when  $\pi = 0.5$  turning into  $\beta_i 0.25$  (i.e., dividing by 4).



# Binomial GLM - Divide by 4 rule



# Binomial GLM - Predicted probabilities

In a similar way we can use the inverse logit function to find the predicted probability specific values of  $x$ . For example, the difference between  $p(y = 1|x = 2.5) - p(y = 1|x = 5)$  can be calculated using the model equation:

- $\text{logit}^{-1}p(y = 1|x = 2.5) = \frac{e^{\beta_0 + \beta_1 2.5}}{1 + e^{\beta_0 + \beta_1 2.5}}$
- $\text{logit}^{-1}p(y = 1|x = 5) = \frac{e^{\beta_0 + \beta_1 5}}{1 + e^{\beta_0 + \beta_1 5}}$
- $\text{logit}^{-1}p(y = 1|x = 5) - \text{logit}^{-1}p(y = 1|x = 2.5)$

```
coefs <- coef(fit)
plogis(coefs[1] + coefs[2]*5) - plogis(coefs[1] + coefs[2]*2.5)
```

```
## (Intercept)
## 0.2237369
```

# Binomial GLM - Predicted probabilities

In R we can use directly the `predict()` function with the argument `type = "response"` to return the predicted probabilities instead of the logits:

```
preds <- predict(fit, newdata = list(x = c(2.5, 5)), type = "response")
preds
```

```
##           1           2
## 0.0329886 0.2567255
```

```
preds[2] - preds[1]
```

```
##           2
## 0.2237369
```

# Binomial GLM - Predicted probabilities

I have written the `epredict()` function that extend the `predict()` function giving some useful messages when computing predictions. you can use it with every model and also with multiple predictors.

```
epredict(fit, values = list(x = c(2.5, 5)), type = "response")
```

```
## y ~ -5.693 + 0.926*c(2.5, 5)
```

```
## [1] 0.0329886 0.2567255
```

# Binomial GLM - Marginal effects

Marginal effects can be considered very similar to the **divide by 4 rule**. A particularly useful type of marginal effect is the **average marginal effect**. While the **divide by 4 rule** estimate the **maximal** difference (in probability scale) according to  $x$ , the **average marginal effect** is the average of all slopes (i.e., derivatives) interpreted as the average change in probability scale across all unit increases in  $x$ .

```
# calculate the derivative
calc_deriv <- function(b0, b1, x){
  (b1 * exp(b0 + b1 * x)) / (1 + (exp(b0 + b1 * x)))^2
}

coefs <- coef(fit)
dd <- calc_deriv(coefs[1], coefs[2], dat$x)
mean(dd)
```

```
## [1] 0.09338663
```

# Binomial GLM - Marginal effects

More efficiently we can do the same using the `margins` package in R:

```
mrg <- margins::margins(fit)
summary(mrg)
```

```
## factor    AME    SE      z      p lower upper
##      x 0.0934 0.0031 29.9339 0.0000 0.0873 0.0995
```

## Binomial GLM - Inference

# Binomial GLM - Wald tests

The basic approach when doing inference with GLM is interpreting the Wald test of each model coefficients. The Wald test is calculated as follows:

$$z = \frac{\beta_i - \beta_0}{\sigma_{\beta_i}}$$

Calculating the p-value based on a standard normal distribution.



# Binomial GLM - Wald-type confidence intervals

Wald-type confidence interval (directly from model summary), where  $\Phi$  is the cumulative Gaussian function `qnorm()`:

$$95\%CI = \hat{\beta} \pm \Phi(\alpha/2)SE_{\beta}$$

```
(summ <- data.frame(summary(fit)$coefficients))
```

```
##           Estimate Std..Error  z.value    Pr...z..  
## (Intercept) -0.6632942  0.2985407 -2.221788 0.0262976067  
## tv_shows     1.4170660  0.4254859  3.330465 0.0008670096
```

```
# 95% confidence interval
```

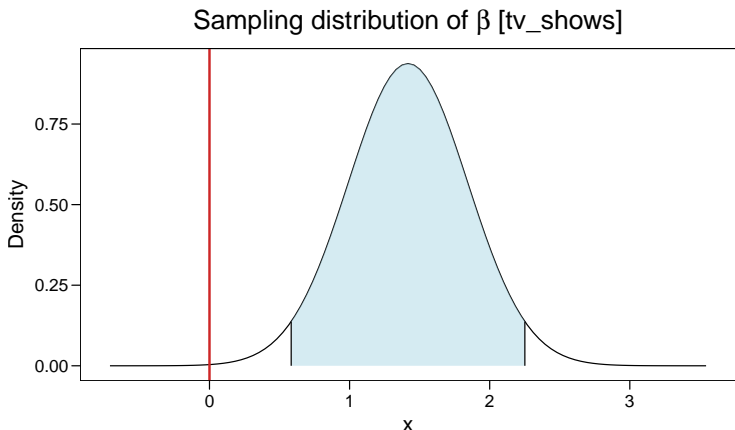
```
summ$Estimate + qnorm(c(0.025, 0.95))*summ$Std..Error
```

```
## [1] -1.248423  2.116928
```

# Binomial GLM - Wald-type confidence intervals

You can also use the `plot_param()` function to represent the sampling distribution and the confidence interval:

```
plot_param(fit, "tv_shows", ci = "z") + mytheme()
```



# Binomial GLM - Profile likelihood confidence intervals

The computation is a little bit different and they are not always symmetric:

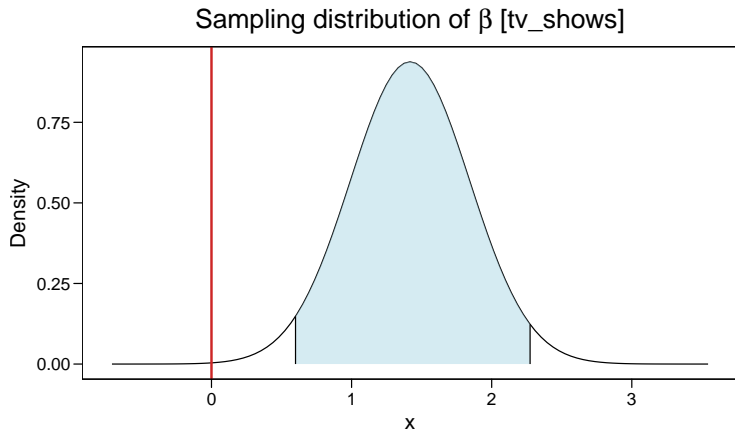
```
# profile likelihood, different from wald type  
confint(fit)
```

```
##              2.5 %      97.5 %  
## (Intercept) -1.2711814 -0.09226204  
## tv_shows      0.5998267  2.27401606
```

# Binomial GLM - Profile likelihood confidence intervals

Again we can use the `plot_param()` function:

```
plot_param(fit, "tv_shows", ci = "profile") + mytheme()
```



# Binomial GLM - Confidence intervals

When calculating confidence intervals it is important to consider the link function. In the same way as we compute the inverse logit function on the parameter value, we could revert also the confidence intervals.

**IMPORTANT, do not apply the inverse logit on the standard error and then compute the confidence interval.**

```
fits <- broom::tidy(fit) # extract parameters as dataframe
fits
```

```
## # A tibble: 2 x 5
##   term          estimate std.error statistic  p.value
##   <chr>          <dbl>    <dbl>    <dbl>    <dbl>
## 1 (Intercept)   -1.52     0.368    -4.12 0.0000380
## 2 tv_shows       2.01     0.469     4.27 0.0000193
```

# Binomial GLM - Confidence intervals

```
b <- fits$estimate[2]
se <- fits$std.error[2]

# correct, wald-type confidence intervals
c(b = exp(b), lower = exp(b - 2*se), upper = exp(b + 2*se))

##          b      lower      upper
## 7.432749  2.906586 19.007090
# correct, likelihood based confidence intervals
exp(confint(fit, "tv_shows"))

##      2.5 %      97.5 %
## 3.062659 19.530409
# wrong wald type
c(b = exp(b), lower = exp(b) - 2*exp(se), upper = exp(b) + 2*exp(se))

##          b      lower      upper
## 7.432749  4.234493 10.631004
```

# Binomial GLM - Confidence intervals

The same principle holds for predicted probabilities. First compute the intervals on the logit scale and then transform-back on the probability scale:

```
fits <- dat |>
  select(tv_shows) |>
  distinct() |>
  add_predict(fit, se.fit = TRUE)

fits$p <- plogis(fits$fit)
fits$lower <- plogis(with(fits, fit - 2*se.fit))
fits$upper <- plogis(with(fits, fit + 2*se.fit))

fits
```

```
## # A tibble: 2 x 7
##   tv_shows  fit se.fit residual.scale    p lower upper
##   <dbl>   <dbl> <dbl>         <dbl> <dbl> <dbl> <dbl>
## 1     1     0.490 0.291             1 0.620 0.477 0.745
## 2     0    -1.52 0.368             1 0.180 0.0951 0.314
```

# Binomial GLM - Anova

With multiple predictors, especially with categorical variables with more than 2 levels, we can compute the an anova-like analysis individuating the effect of each predictor. In R we can do this using the `car::Anova()` function. Let's simulate a model with a 2x2 interaction:

```
##   id x1 x2 y
## 1  1  a  c  1
## 2  2  a  d  0
## 3  3  b  c  0
## 4  4  b  d  0
## 5  5  a  c  1
## 6  6  a  d  0
```

We can fit the most complex model containing the two main effects and the interaction<sup>2</sup>:

```
fit_max <- glm(y ~ x1 + x2 + x1:x2, data = dat, family = binomial(link = "logit")) # same as x1 * x2
```

---

<sup>2</sup>I set the contrasts for the two factors as `contr.sum()/2` that are required for a proper analysis of factorial designs



# Binomial GLM - Anova

```
summary(fit_max)
```

```
##
## Call:
## glm(formula = y ~ x1 + x2 + x1:x2, family = binomial(link = "logit"),
##      data = dat)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.066  -1.011  -0.459   1.293   2.146
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.0151     0.2279  -4.454 8.44e-06 ***
## x11           0.5724     0.4559   1.256 0.20924
## x21           1.3565     0.4559   2.976 0.00292 **
## x11:x21       -0.8704     0.9117  -0.955 0.33973
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 144.87  on 119  degrees of freedom
## Residual deviance: 133.54  on 116  degrees of freedom
## AIC: 141.54
##
## Number of Fisher Scoring iterations: 4
```

# Binomial GLM - Anova

Then using `car::Anova()`. For each effect we have the  $\chi^2$  statistics and the associated p-value. The null hypothesis is that the specific factor did not contribute in reducing the residual deviance.

```
car::Anova(fit_max)
```

```
## Analysis of Deviance Table (Type II tests)
```

```
##
```

```
## Response: y
```

```
##      LR Chisq Df Pr(>Chisq)
```

```
## x1      1.0943  1  0.295529
```

```
## x2      9.3856  1  0.002187 **
```

```
## x1:x2    0.9403  1  0.332195
```

```
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

# Binomial GLM - Model comparison

The table obtained with `car::Anova()` is essentially a model comparison using the Likelihood Ratio test. This can be done using the `anova(...)` function.

$$D = 2(\log(\mathcal{L}_{full}) - \log(\mathcal{L}_{reduced}))$$

$$D \sim \chi^2_{df_{full}-df_{reduced}}$$

# Binomial GLM - Model comparison

To better understanding, the `x2` effect reported in the `car::Anova()` table is a model comparison between a model with  $y \sim x1 + x2$  and a model without `x2`. The difference between these two model is the unique contribution of `x2` after controlling for `x1`:

```
fit <- glm(y ~ x1 + x2, data = dat, family = binomial(link = "logit"))
fit0 <- glm(y ~ x1, data = dat, family = binomial(link = "logit"))

anova(fit0, fit, test = "LRT") # ~ same as car::Anova(fit_max)
```

```
## Analysis of Deviance Table
##
## Model 1: y ~ x1
## Model 2: y ~ x1 + x2
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1         118      143.86
## 2         117      134.48 1    9.3856 0.002187 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

# Binomial GLM - Model comparison

The model comparison using `anova()` (i.e., likelihood ratio tests) is limited to nested models thus models that differs only for one term. For example:

```
fit1 <- glm(y ~ x1, data = dat, family = binomial(link = "logit"))
fit2 <- glm(y ~ x2, data = dat, family = binomial(link = "logit"))
fit3 <- glm(y ~ x1 + x2, data = dat, family = binomial(link = "logit"))
```

`fit1` and `fit2` are non-nested because we have the same number of predictors (thus degrees of freedom). `fit3` and `fit1/fit2` are nested because `fit3` is more complex and removing one term we can obtain the less complex models.

# Binomial GLM - Model comparison

```
anova(fit1, fit2, test = "LRT") # do not works properly
```

```
## Analysis of Deviance Table
##
## Model 1: y ~ x1
## Model 2: y ~ x2
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1         118       143.86
## 2         118       135.57  0    8.2914
```

```
anova(fit1, fit3, test = "LRT") # same anova(fit2, fit3)
```

```
## Analysis of Deviance Table
##
## Model 1: y ~ x1
## Model 2: y ~ x1 + x2
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1         118       143.86
## 2         117       134.48  1    9.3856 0.002187 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

# Binomial GLM - Information Criteria

As for standard linear models I can use the Akaike Information Criteria (AIC) or the Bayesian Information Criteria (BIC) to compare non-nested models. The downside is not having a properly hypothesis testing setup.

```
data.frame(BIC(fit1, fit2, fit3)) |>  
  arrange(BIC)
```

```
##      df      BIC  
## fit2  2 145.1455  
## fit3  3 148.8387  
## fit1  2 153.4369
```

```
data.frame(AIC(fit1, fit2, fit3)) |>  
  arrange(AIC)
```

```
##      df      AIC  
## fit2  2 139.5705  
## fit3  3 140.4763  
## fit1  2 147.8619
```

## Binomial GLM - Plotting effects



# Binomial GLM - Marginal effects

When plotting a binomial GLM the most useful way is plotting the marginal probabilities and standard errors/confidence intervals for a given combination of predictors. Let's make an example for:

- simple GLM with 1 categorical/numerical predictor
- GLM with 2 numerical/categorical predictors
- GLM with interaction between numerical and categorical predictors

# Binomial GLM - Marginal effects

A general workflow could be:

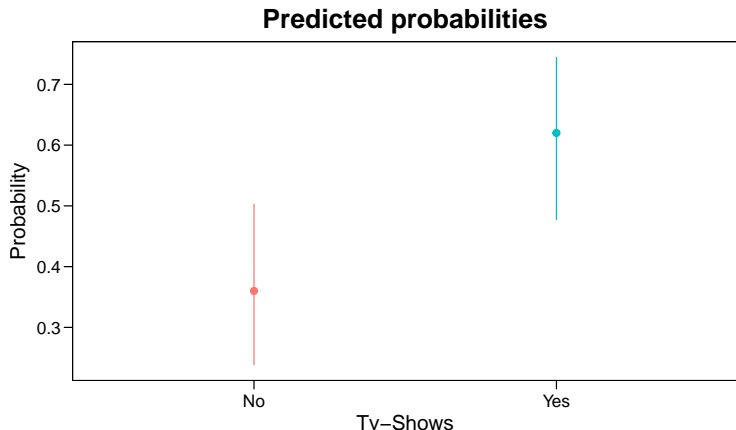
- fit the model
- use the `predict()` function giving the grid of values on which computing predictions
- calculating the confidence intervals
- plotting the results

Everything can be simplified using some packages to perform each step and returning a plot:

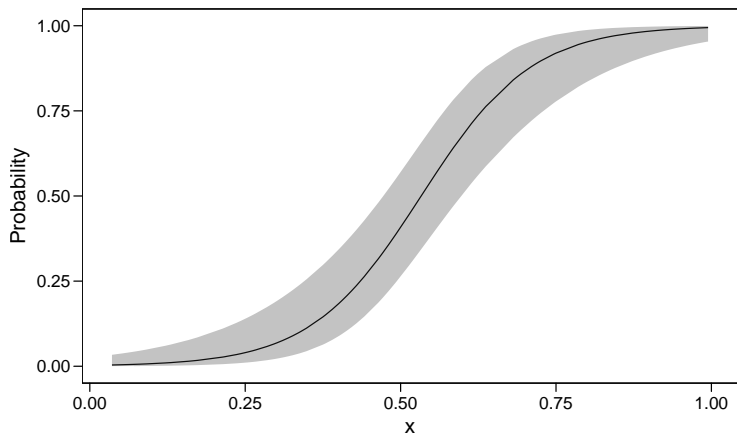
- `allEffects()` from the `effects()` package (return a base R plot)
- `ggeffect()` from the `ggeffect()` package (return a `ggplot2` object)
- `plot_model` from the `sjPlot` package (similar to `ggeffect()`)

# Binomial GLM - 1 categorical predictor

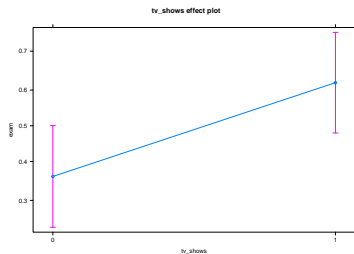
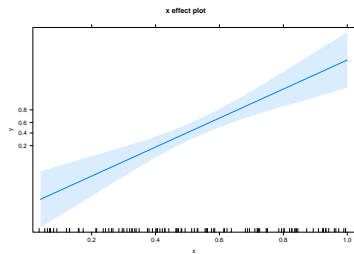
In this situation we can just plot the marginal probabilities for each level of the categorical predictor. Plotting our exam dataset:



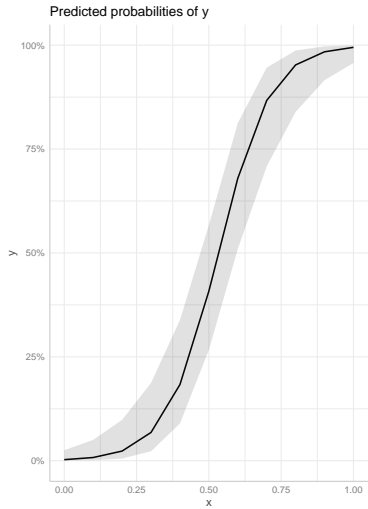
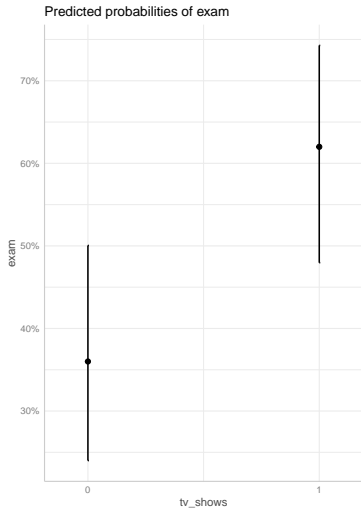
# Binomial GLM - 1 numerical predictor



# Binomial GLM - allEffects()

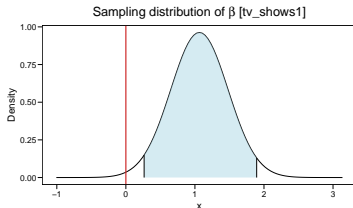


# Binomial GLM - `ggeffect()/plot_model()`



# Binomial GLM - Plotting coefficients

Sometimes could be useful to plot the estimated sampling distribution of a coefficient. For example, we can plot the `tv_shows` effect on our example. I've written the `plot_param()` function that directly create a basic-plot given the model and the coefficient name. The plot highlight the null value and the 95% Wald confidence interval.



## GLM - Diagnostic



# GLM - Diagnostic

The diagnostic for GLM is similar to standard linear models. Some areas are more complicated for example residual analysis and goodness of fit.

We will see:

- Deviance
- $R^2$
- Residuals
  - Types of residuals
  - Residual deviance
- Classification accuracy

# Likelihood

The likelihood is the joint probability of the observed data viewed as a function of the parameters of a statistical model.

$$\log \mathcal{L}(\mu|x) = \sum_{i=1}^n \log(\mu|x_i)$$

```
x <- rnorm(10) # data from normal distribution

# data
x

## [1] -1.5324038  1.9535253  1.2066729 -1.2474664 -0.9444257 -0.7751357
## [7]  0.4442473  0.8424928  0.2315650 -1.2960886

# the model is a normal distribution with mu = 0 and sd = 1
dnorm(x, 0, 1)

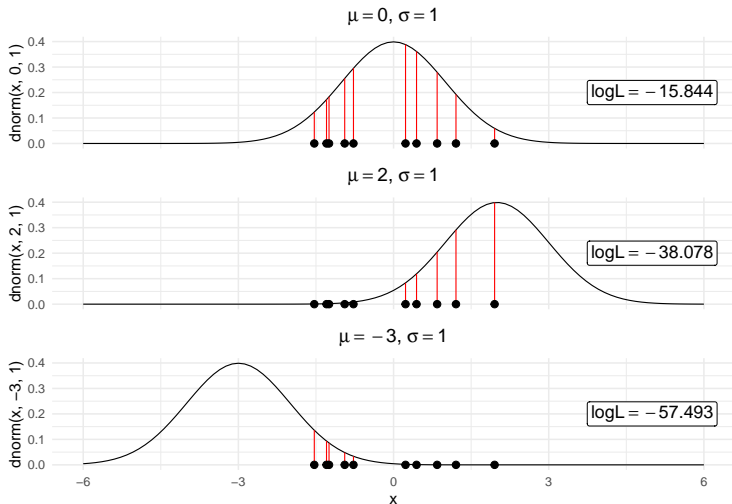
## [1] 0.12330810 0.05918607 0.19263303 0.18322786 0.25540406 0.29542030
## [7] 0.36145550 0.27975652 0.38838829 0.17224086

# log likelihood
sum(log(dnorm(x, 0, 1)))

## [1] -15.84446
```

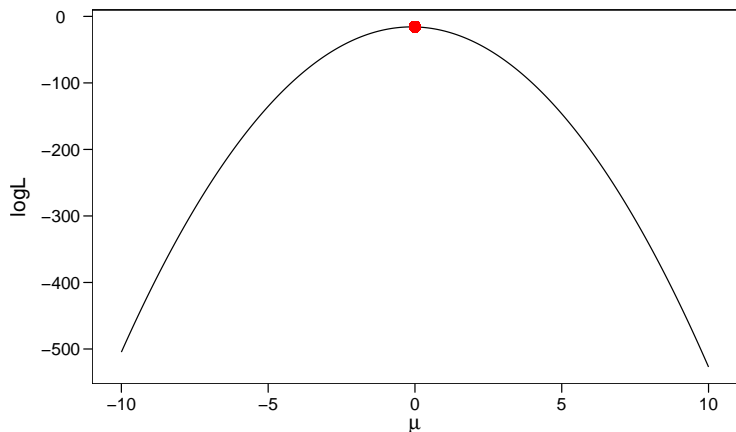
# Likelihood

By summing the logarithm all the red segments we obtain the log likelihood of the model given the observed data.



# Likelihood

In the previous slide we tried only 3 values for the mean. Let's image to calculate the log likelihood for several different means. The parameter with that is associated with highest likelihood is called the maximum likelihood estimator. In fact, the  $\mu = 0$  is associated with the highest likelihood.



# Deviance

The (residual) deviance in the context of GLM can be considered as the distance between the current model and a perfect model (often called *saturated model*).

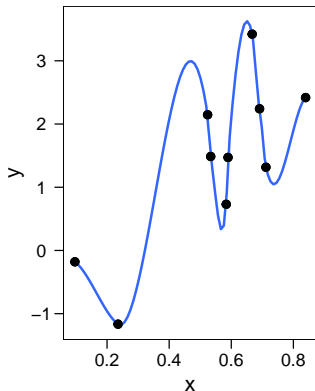
$$D_{res} = -2[\log(\mathcal{L}_{current}) - (\log \mathcal{L}_{sat})]$$

Where  $\mathcal{L}$  is the likelihood of the considered model (see the previous slides). Clearly, the lower the deviance, the closer the current model to the perfect model suggesting a good fit.

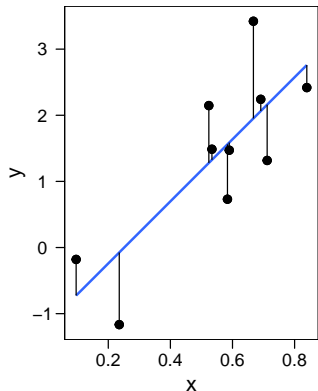
# Deviance - Saturated model

The saturated model is a model where each observation  $x_i$  has a parameter.

**Saturated model  $y \sim id$**



**Current model  $y \sim x$**



# Deviance - Null model

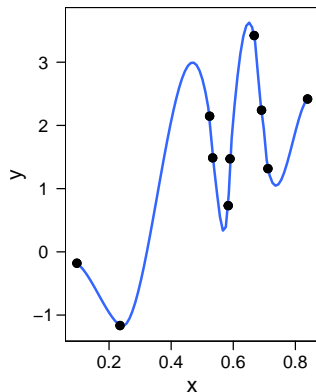
Another important quantity is the *null deviance* that is expressed as the distance between the *null model* and the *saturated model*.

$$D_{null} = -2[\log(\mathcal{L}_{null}) - (\log \mathcal{L}_{sat})]$$

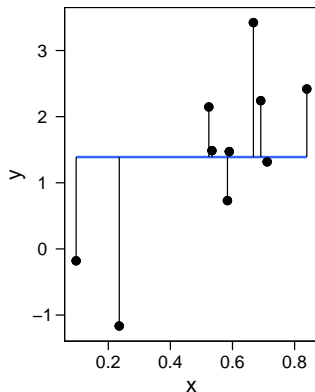
The *null deviance* can be interpreted as the maximal deviance because is estimated using a model without predictors. A good model will have a residual deviance lower than the null model.

# Deviance - Null model

**Saturated model  $y \sim id$**



**Null model  $y \sim 1$**





# Deviance - Likelihood Ratio Test

When we perform a likelihood ratio test (see previous slides) we are essentially comparing the residual deviance (or the likelihood) of two models.

```
fit_null <- glm(y ~ 1, data = dat, family = binomial(link = "logit"))  
fit_current <- glm(y ~ x, data = dat, family = binomial(link = "logit"))
```

# Deviance - Likelihood Ratio Test

With the null model clearly the residual deviance is the same as the null deviance because we are not using predictors to reduce the deviance.

```
summary(fit_null)
```

```
##
## Call:
## glm(formula = y ~ 1, family = binomial(link = "logit"), data = dat)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.044  -1.044  -1.044   1.317   1.317
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.3228     0.2865  -1.126    0.26
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 68.029  on 49  degrees of freedom
## Residual deviance: 68.029  on 49  degrees of freedom
## AIC: 70.029
##
## Number of Fisher Scoring iterations: 4
```

# Deviance - Likelihood Ratio Test

If the predictor  $x$  is useful in explaining  $y$  the residual deviance will be reduced compared to the null (overall deviance).

```
summary(fit_current)

##
## Call:
## glm(formula = y ~ x, family = binomial(link = "logit"), data = dat)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -2.9121  -0.4071  -0.2063   0.6259   1.7336
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -5.045      1.470  -3.432 0.000599 ***
## x              9.535      2.659   3.586 0.000336 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 68.029  on 49  degrees of freedom
## Residual deviance: 36.745  on 48  degrees of freedom
## AIC: 40.745
##
## Number of Fisher Scoring iterations: 6
```

# Deviance - Likelihood Ratio Test

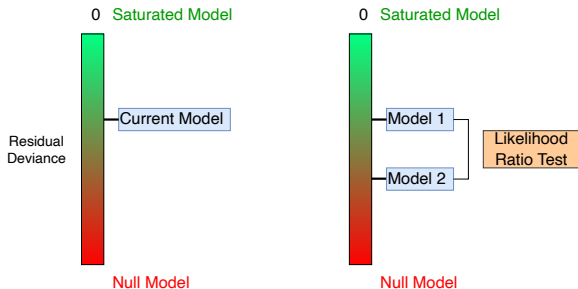
Comparing the two models we can understand if the deviance reduction can be considered statistically significant.

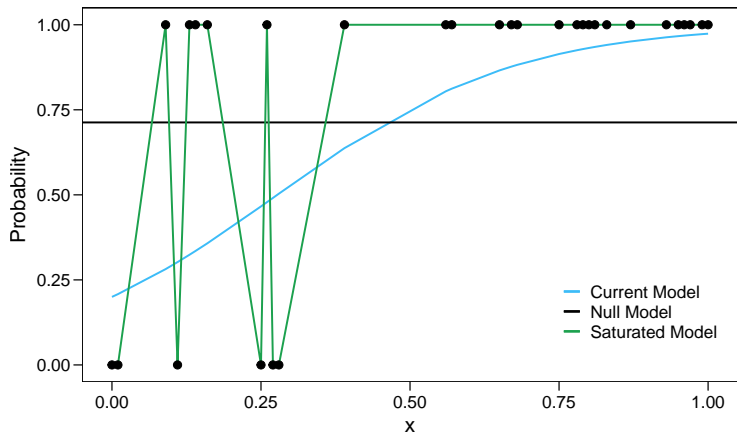
```
anova(fit_null, fit_current, test = "LRT")
```

```
## Analysis of Deviance Table
##
## Model 1: y ~ 1
## Model 2: y ~ x
##   Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
## 1         49      68.029
## 2         48      36.745  1    31.284 2.229e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Notice that the difference between the two residual deviances is the test statistics that is distributed as a  $\chi^2$  with  $df = 1$ .

# Deviance - Likelihood Ratio Test





<sup>3</sup>Adapted from <https://bookdown.org/egarpor/SSS2-UC3M/logreg-deviance.html>

# Deviance - Example #extra

Let's fit the three models:

```
# null
fit0 <- glm(exam ~ 1, data = dat, family = binomial(link = "logit"))
# current
fit <- glm(exam ~ tv_shows, data = dat, family = binomial(link = "logit"))
# saturated
fits <- glm(exam ~ 0 + id, data = dat, family = binomial(link = "logit"))
```

# Deviance - Example #extra

We can calculate the residual deviance:

```
-2*(logLik(fit) - logLik(fits))
```

```
## 'log Lik.' 126.3001 (df=2)
```

```
deviance(fit)
```

```
## [1] 126.3001
```

We can calculate the null deviance:

```
-2*(logLik(fit0) - logLik(fit))
```

```
## 'log Lik.' 4.383571 (df=1)
```

```
deviance(fit0)
```

```
## [1] 130.6836
```



# Binomial GLM - $R^2$

Compared to the standard linear regression, there are multiple ways to calculate an  $R^2$  like measure for GLMs and there is no consensus about the most appropriate method. There are some useful resources:

- <https://stats.oarc.ucla.edu/other/mult-pkg/faq/general/faq-what-are-pseudo-r-squareds/>

To note, some measures are specific for the binomial GLM while other measures can be applied also to other GLMs (e.g., the poisson)

# Binomial GLM - $R^2$

We will see:

- McFadden's pseudo- $R^2$  (for GLMs in general)
- Nagelkerke's  $R^2$  (for GLMs in general)
- Tjur's  $R^2$  (only for binomial/binary models)

# McFadden's pseudo- $R^2$

The McFadden's pseudo- $R^2$  compute the ratio between the log-likelihood of the intercept-only (i.e., null) model and the current model[3]:

$$R^2 = 1 - \frac{D_{current}}{D_{null}}$$

There is also the adjusted version that take into account the number of parameters of the model. In R can be computed manually or using the `performance::r2_mcfadden()`:

```
performance::r2_mcfadden(fit)
```

```
## # R2 for Generalized Linear Regression
##      R2: 0.034
##      adj. R2: 0.018
```

# Tjur's $R^2$

This measure is the easiest to interpret and calculate but can only be applied for binomial binary models [4]. Is the absolute value of the difference between the proportions of correctly classifying  $y = 1$  and  $y = 0$  from the model:

$$\begin{aligned}\pi_1 &= p(y_i = 1 | \hat{y}_i = 1) \\ \pi_2 &= p(y_i = 0 | \hat{y}_i = 0) = 1 - \pi_1 \\ R^2 &= |\pi_1 - \pi_2|\end{aligned}$$

```
performance::r2_tjur(fit2)
```

```
## Tjur's R2  
## 0.6070024
```

# Binomial GLM - Residuals

As for standard linear models there are different types of residuals:

- raw (response) residuals
- pearson residuals
- standardized pearson residuals
- deviance residuals

# Raw Residuals

Raw residuals, also called response residuals are the simplest type of residuals. They are calculated as in standard regression as:

$$r_i = y_i - \hat{y}_i$$

Where  $\hat{y}_i$  are the fitted values where the inverse of the link function has been applied.

In R:

```
# equivalent to residuals(fit, type = "response")
ri <- fit$y - fitted(fit)
ri[1:5]
```

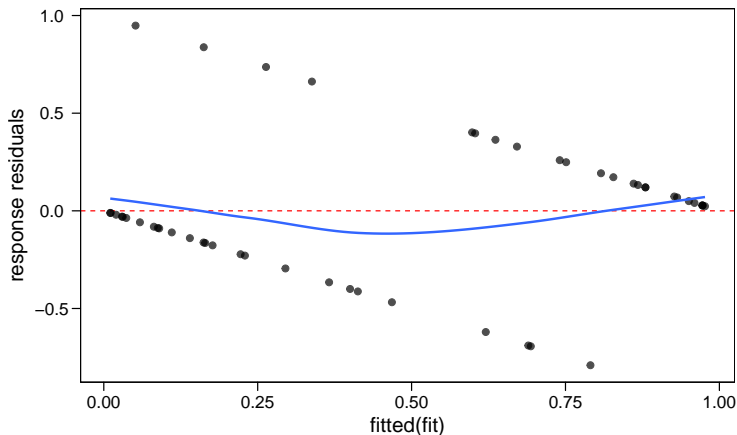
```
##           1           2           3           4           5
## -0.4680320  0.1392685  0.3288338 -0.6895866 -0.1647523
```

The problem is that in GLMs the mean and the variance of the distribution are not independent, creating problems in residual analysis.

# Raw Residuals

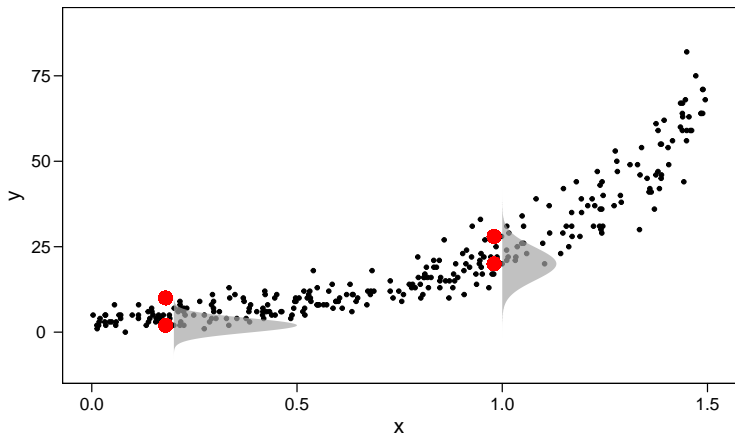
We can use the custom function `plot_residuals()` to plot different type of residuals against fitted values:

```
plot_resid(fit, type = "response") + mytheme()
```



# Why raw residuals are problematic?

This plot<sup>4</sup> shows an example with the same residual for two different  $x$  values on a Poisson GLM. Beyond the model itself, the same residual can be considered as extreme for low  $x$  values and plausible for high  $x$  values:



<sup>4</sup>Adapted from Dunn (2018), Fig. 8.1



# Binomial GLM - Binned (raw) Residuals

Gelman and colleagues [2] proposed a type of residuals called **binned residuals** to solve the problem of the previous plot for Binomial GLMs:

- divide the fitted values into  $n$  bins. The number is arbitrary but we need each bin to have enough observation to compute a reliable average
- calculate the average fitted value and residual for each bin
- for each bin we can compute the standard error as  $SE = \frac{\hat{p}_j(1-\hat{p}_j)}{n_j}$  where  $\hat{p}_j$  is the average fitted probability and  $n_j$  is the number of observation in the bin  $j$
- Then we can plot each bin and the confidence intervals (e.g., as  $\pm 2 * SE$ ) where  $\sim 95\%$  of binned residuals should be within the CI if the model is true

# Binomial GLM - Binned (raw) Residuals

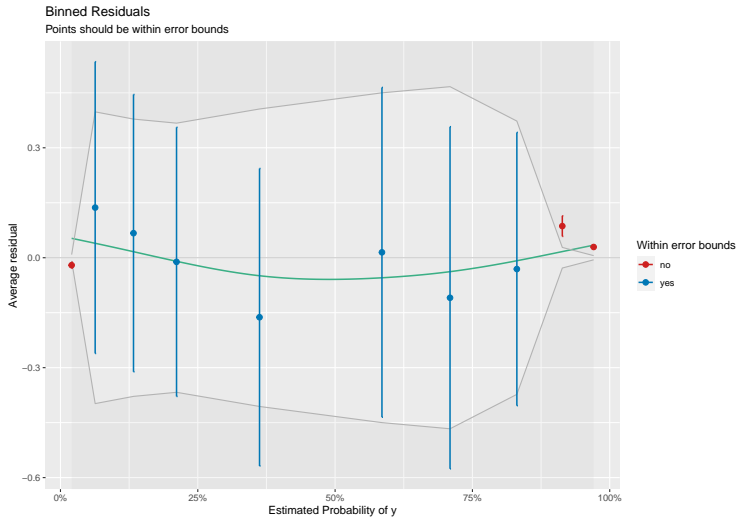
We can use the `performance::binned_residuals(model = , n_bins = )` function to automatically create and plot the binned residuals:

```
bres <- performance::binned_residuals(fit, n_bins = 10) # 10 is the default
head(data.frame(bres))
```

```
##           xbar      ybar  n      x.lo      x.hi      se      ci_range
## 1 0.02034898 -0.02034898 5 0.01049968 0.03076902 0.008487489 0.004330431
## 2 0.06318050 0.13681950 5 0.03652146 0.08771356 0.398053007 0.203092001
## 3 0.13296139 0.06703861 5 0.09002218 0.16235891 0.378358525 0.193043611
## 4 0.21130882 -0.01130882 5 0.16475227 0.26352505 0.367224937 0.187363104
## 5 0.36239336 -0.16239336 5 0.29508386 0.41270883 0.405879048 0.207084952
## 6 0.58529230 0.01470770 5 0.46803198 0.63625848 0.449946460 0.229568739
##           CI_low      CI_high group
## 1 -0.02883647 -0.01186149      no
## 2 -0.26123351 0.53487250      yes
## 3 -0.31131991 0.44539714      yes
## 4 -0.37853376 0.35591611      yes
## 5 -0.56827241 0.24348569      yes
## 6 -0.43523876 0.46465416      yes
```

# Binomial GLM - Binned (raw) Residuals

```
plot(bres)
```



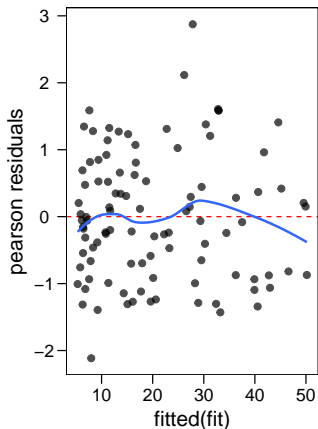
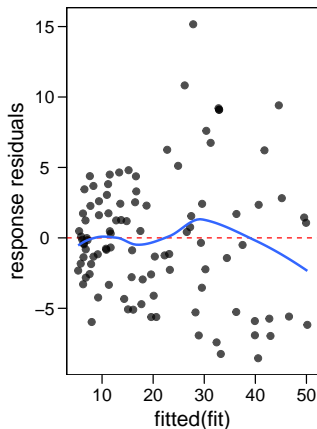
# Pearson residuals

Pearson residuals are raw residuals divided by the standard deviation of each residual. Given that the mean-variance relationship of GLMs, dividing by the standard deviation solve the problem of slide 104. The denominator can be calculated just using the appropriate variance formula for that specific GLM.

$$r_i = \frac{y_i - \hat{y}_i}{\sqrt{V(\hat{y}_i)}}$$

# Pearson vs Raw residuals

We can see the difference for a Poisson model<sup>5</sup> when using raw vs pearson residuals. The non-constant variance is controlled on the right.



<sup>5</sup>We are using a Poisson model only because the residual pattern is more clear compared to a binomial model

# Binomial GLM - Pearson residuals

For the Binomial (Bernoulli) GLM, the variance is calculated as  $\hat{\pi}(1 - \hat{\pi})$  where  $\hat{\pi}$  is the residual value.

$$r_i = \frac{y_i - \hat{y}_i}{\sqrt{\hat{y}_i(1 - \hat{y}_i)}}$$

```
# equivalent to residuals(fit, type = "pearson")
yi <- fitted(fit)
ri_pearson <- ri / sqrt(yi * (1 - yi))
ri_pearson[1:5]
```

```
##           1           2           3           4           5
## -0.9379831  0.4022468  0.6999599 -1.4904733 -0.4441279
```

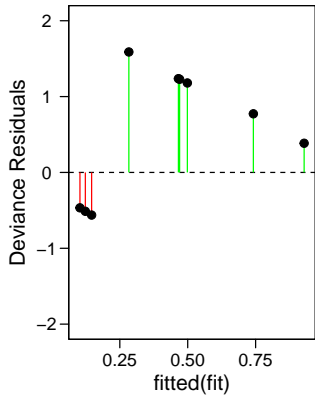
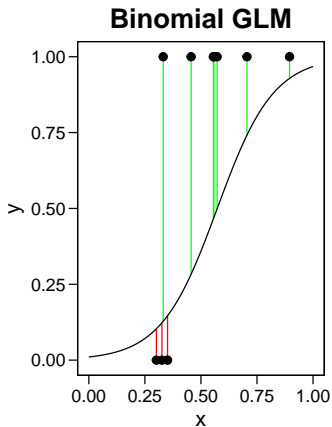
# Deviance residuals

Deviance residuals are based on the residual deviance that we defined before. In fact, the residual deviance was just the sum of squared deviance residuals.

$$r_i = \text{sign}(y_i - \hat{y}_i) \sqrt{-2[\log \mathcal{L}y_{i_{\text{current}}} - \log \mathcal{L}y_{i_{\text{saturated}}}]}$$

Where *sign* is the sign of the raw residual *i*. For the Bernoulli model the  $\log \mathcal{L}y_{i_{\text{saturated}}}$  is always 0.

# Binomial GLM - Deviance Residuals





# Binomial GLM - Deviance Residuals #extra

To calculate and demonstrate manually the deviance residuals we can compute them manually in R:

```
# residual(fit, type = "deviance")
yhat <- fitted(fit) # fitted
y <- fit$y # actual values

# the likelihood is dbinom
ri_dev <- sign(y - yhat) * sqrt(-2*(log(dbinom(y, 1, yhat)) - log(dbinom(y, 1, y))))

# notice that the log lik of the saturated model log(dbinom(y, 1, y)) is 0
log(dbinom(y, 1, y))
```

```
## 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ri_dev[1:5]
```

```
##          1          2          3          4          5
## -0.2370588  0.3033530 -1.0310244  0.2331303 -0.8429485
residuals(fit, type = "deviance")[1:5]
```

```
##          1          2          3          4          5
## -0.2370588  0.3033530 -1.0310244  0.2331303 -0.8429485
```

# Quick recap about hat values in linear regression

The **hat matrix**  $H$  is calculated as  $H = X(X^\top X)^{-1}X^\top$  is a  $n \times n$  matrix where  $n$  is the number of observations. The diagonal of the  $H$  matrix contained the hat values or leverages.

The  $i^{th}$  leverage score ( $h_{ii}$ ) is interpreted as the weighted distance between  $x_i$  and the mean of  $x_i$ 's. In practical terms is the  $i^{th}$  observed value influence the  $i^{th}$  fitted value. An high leverage suggest that the observation is far from the mean of predictors and have an high influence on the fitted values.

- $h_{ii}$  ranges between 0 and 1
- The sum of all  $h_{ii}$  values is the number of parameters  $p$
- As a rule of thumb, an observation have an high leverage if  $h_{ii} > 2\bar{h}$  where  $\bar{h}$  is the average hat value

# Quick recap about hatvalues in linear regression

For a simple linear regression ( $y \sim x$ ) the hat values are calculated as:

$$h_i = \frac{1}{n} + \frac{(X_i - \bar{X})^2}{\sum_{j=1}^n (X_j - \bar{X})^2}$$

In R the function `hatvalues()` return the diagonal of the  $H$  matrix for `glm` and `lm`:

```
hatvalues(fit)[1:10]
```

```
##           1           2           3           4           5           6           7
## 0.02874015 0.03492581 0.03968557 0.02782301 0.04357230 0.04652508 0.04435804
##           8           9          10
## 0.04012613 0.03948651 0.04743780
```

To note, for GLM and multiple regression in general, the equation is different and more complicated but the interpretation and the R implementation is the same.

# Standardized Residuals

Both the response, pearson and deviance residuals can be considered as raw residuals. We can standardize residuals by dividing for the standard error computed with the hat values. In this way, the distribution will be approximately normal with  $\mu = 0$  and  $\sigma = 1$ .

$$r_{s_i} = \frac{r_i}{\sqrt{(1 - \hat{h}_{ii})}}$$

Where  $r_i$  can be raw, pearson or deviance residuals.

# Standardized Residuals

In R they can be extracted using `rstandard()`:

```
rstandard(fit, type = "pearson")[1:5]
```

```
##           1           2           3           4           5  
## -0.1712897  0.2208858 -0.8546823  0.1683326 -0.6678440
```

```
rstandard(fit, type = "deviance")[1:5]
```

```
##           1           2           3           4           5  
## -0.2405406  0.3087934 -1.0521126  0.2364427 -0.8619359
```

# Binomial GLM Standardized Residuals #extra

We can try to manually calculate the residuals to better understand.

```
yhat <- fitted(fit) # fitted
yi <- fit$y # observed
hi <- hatvalues(fit) # diagonal of the hat matrix
```

```
# pearson residuals
pi <- (yi - yhat) / sqrt(yhat * (1 - yhat))
```

```
# standardized
pis <- pi / sqrt(1 - hi)
pis[1:5]
```

```
##           1           2           3           4           5
## -0.1712897  0.2208858 -0.8546823  0.1683326 -0.6678440
rstandard(fit, type = "pearson")[1:5]
```

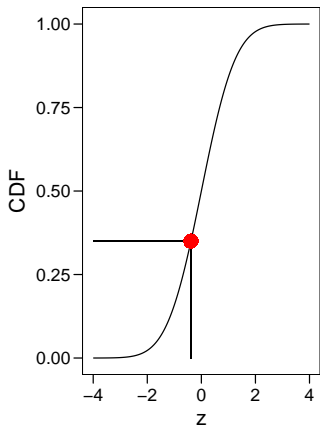
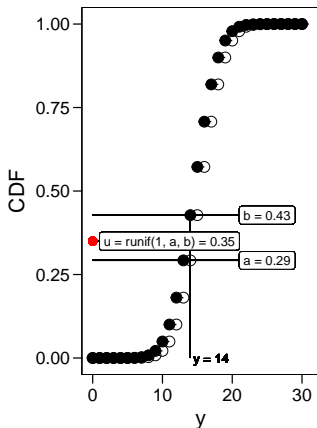
```
##           1           2           3           4           5
## -0.1712897  0.2208858 -0.8546823  0.1683326 -0.6678440
# deviance (for binomial GLM the loglik of the saturated model is 0)
di <- sign(yi - yhat) * sqrt(-2*log(dbinom(y, 1, yhat)))
```

```
# standardized
dis <- di / sqrt(1 - hi)
dis[1:5]
```

```
##           1           2           3           4           5
## -0.2405406  0.3087934 -1.0521126  0.2364427 -0.8619359
rstandard(fit, type = "deviance")[1:5]
```

# Quantile residuals #extra

The **quantile residuals** is another proposal for residual analysis. The idea is to map the quantile of the cumulative density function (CDF) of the random component into the CDF of the normal distribution.



# Binomial GLM - Quantile residuals #extra

**Quantile residuals** are very useful especially for Discrete GLMs (binomial and poisson) and are exactly normally distributed (under respected model assumptions) compared to **deviance** and **pearson** residuals [5]. They can be calculated using the `statmod::qresid(fit)` function. Authors suggest to run the function 4 times to disentangle between the randomization and the systematic component.

```
statmod::qresid(fit)[1:5]
```

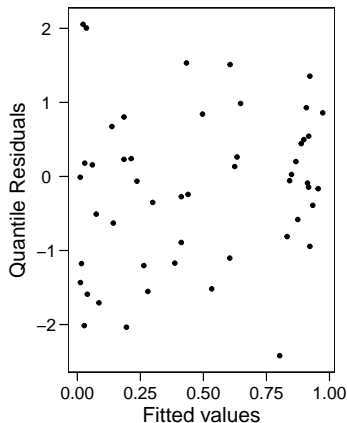
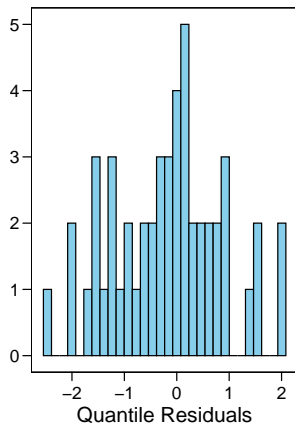
```
## [1] -0.33677932  1.04616589 -0.43878900  0.01511622 -1.07265867
```

```
statmod::qresid(fit)[1:5] # different every time
```

```
## [1]  1.2349067 -0.6689038  0.2171433 -0.9988869 -0.4189008
```



# Binomial GLM - Quantile residuals #extra



# Binomial GLM - Classification accuracy/Error rate

The **error rate** (ER) is defined as the proportion of cases for which the deterministic prediction i.e. guessing  $y_i = 1$  if  $\text{logit}^{-1}(\hat{y}_i) > 0.5$  and guessing  $y_i = 0$  if  $\text{logit}^{-1}(\hat{y}_i) < 0.5$  is wrong. Clearly,  $1 - ER$  is the **classification accuracy**.

I wrote the `error_rate` function that simply compute the error rate of a given model:

# Binomial GLM - Classification accuracy/Error rate

```
error_rate(fit)
```

```
## [1] 0.14
```

We could compare the error rate of a given model with the error rate of the null model or another similar model (with a model comparison approach):

```
fit0 <- update(fit, . ~ -x) # removing the x predictor, now intercept only model  
error_rate(fit0)
```

```
## [1] 0.48
```

```
# the error rate of the null model is ... greater/less than the actual model  
er_ratio = error_rate(fit0)/error_rate(fit)
```

The **error rate** of the null model is 3.429 times greater than the actual model.

# Binomial GLM - ROC analysis #extra

This prediction-based approach can be extended using the logistic regression to compute the receiver operating characteristic analysis. This is a more advanced topic but the idea is to use a binomial regression to make predictions and assess the prediction accuracy. This is commonly used in machine learning where the model is trained on a set of data trying to predict a set of new data.

The ROC curve is a tool to assess the performance of a classifier model (e.g., binomial regression). The idea is to use several different threshold to create the 0/1 predictions (instead of 0.5 as in the previous slides) and find the optimal value.

# Binomial GLM - ROC analysis #extra

Let's use again the 0.5 threshold but computing all the classification metrics. We can just create a 2x2 table with model-based predictions (often called confusion matrix):

```
pi <- ifelse(predict(fit, type = "response") > 0.5, 1, 0)
yi <- fit$y

# confusion matrix
table(pi, yi)
```

```
##      yi
## pi    0  1
##    0 31  7
##    1  4  8
```

# Binomial GLM - ROC analysis #extra

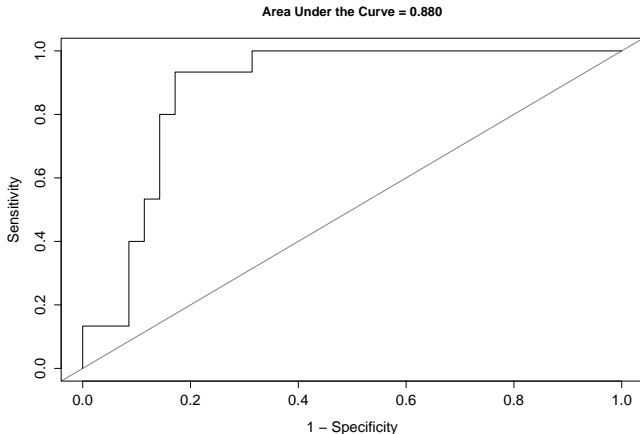
There are several metrics to compute in a confusion matrix (see here). To create a ROC curve we need to calculate the **True Positive Rate** (TPR, also called *sensitivity*) representing the proportion of  $\hat{y} = 1$  when  $y = 1$  using a specific threshold value and the **False Positive Rate** (FPR, also called  $1 - \text{specificity}$ ) representing one minus the proportion of  $\hat{y} = 1$  when  $y = 0$ . We could use the `classify(fit, th = )` function to compute relevant metrics from a fitted model and a given threshold:

```
data.frame(classify(fit, 0.5))
```

```
##   tp tn fp fn      tpr      tnr      fnr      fpr
## 1  8 31  4  7 0.5333333 0.8857143 0.4666667 0.1142857
```

# Binomial GLM - ROC analysis #extra

Then simply using the `classify()` function with multiple thresholds we can have two vectors of TPRs and FPRs and plotting the ROC curve. The Area Under the Curve (AUC) ranges between 0 and 1 (more realistically between 0.5 and 1). The AUC is the ability of the classifier to classify  $y$  values:



# Binomial GLM - Outliers and influential observations

Identification of influential observation and outliers of GLMs is very similar to standard regression models. We will briefly see:

- Studentized residuals
- Cook Distances
- DFBETAs



# Binomial GLM - Cook Distances

The Cook Distance of an observation  $i$  measured the impact of that observation on the overall model fit. If removing the observation  $i$  has an high impact, the observation  $i$  is likely an influential observation. For GLMs they are defined as:

$$D_i = \frac{r_i^2}{\phi p} \frac{h_{ii}}{1 - h_{ii}}$$

Where  $p$  is the number of model parameters,  $r_i$  are the standardized pearson residuals (`rstandard(fit, type = "pearson")`) and  $h_{ii}$  are the hatvalues (leverages).  $\phi$  is the dispersion parameter of the GLM that for binomial and poisson models is fixed to 1 (see Dunn (2018, Table 5.1)) Usually an observation is considered influential if  $D_i > \frac{4}{n}$  where  $n$  is the sample size.

# Binomial GLM - DFBETAs

DFBETAs measure the impact of the observation  $i$  on the estimated parameter  $\beta_j$ :

$$DFBETAS_i = \frac{\beta_j - \beta_{j(i)}}{\sigma_{\beta_{j(i)}}}$$

Where  $i$  denote the parameters and standard error on a model fitted without the  $i$  observation. Usually an observation is considered influential if  $|DFBETAS_i| > \frac{2}{\sqrt{n}}$  where  $n$  is the sample size.

# Binomial GLM - Extracting influence measures

In R we can use the `influence.measures()` function to calculate each influence measure explained before<sup>6</sup>.

```
fit <- update(fit, data = fit$model[1:50, ])
```

```
infl <- influence.measures(fit)$infmtat  
head(infl)
```

##	dfb.1_	dfb.x	dffit	cov.r	cook.d	hat
## 1	-0.11348851	0.16703522	0.21863739	1.086784	0.0110322891	0.06693001
## 2	-0.05079282	0.04641438	-0.05108673	1.071081	0.0005452688	0.02973196
## 3	-0.03356661	0.03135594	-0.03359876	1.065576	0.0002342899	0.02316136
## 4	-0.05321089	0.04847139	-0.05357232	1.071662	0.0006002579	0.03050668
## 5	-0.12572220	0.08964145	-0.15632585	1.058777	0.0055906317	0.03940285
## 6	-0.06346662	0.05702115	-0.06423431	1.073641	0.0008672340	0.03344461

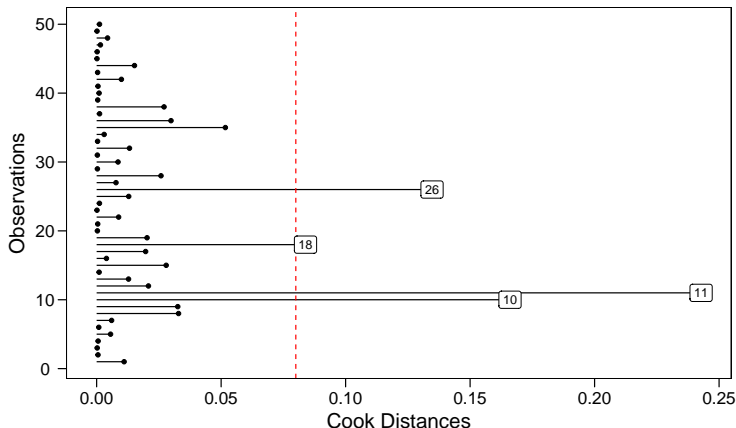
---

<sup>6</sup>The function actually computes also other influence measures, see Dunn (2018, section 8.8.3) for other details

# Binomial GLM - Plotting influence measures

I wrote the `cook_plot()` function to easily plot the cook distances along with the identification of influential observations:

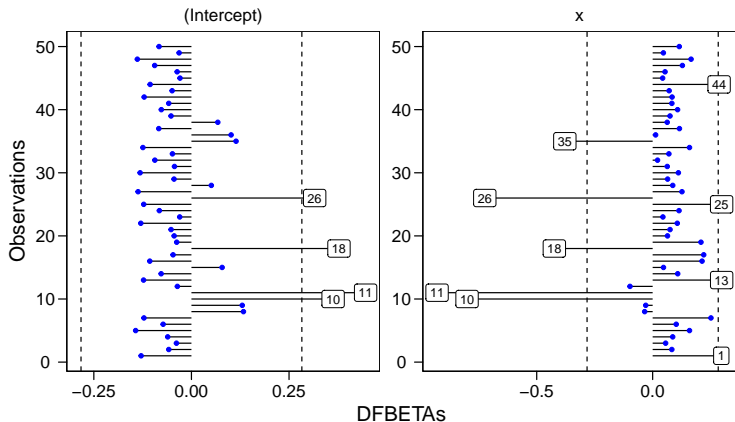
```
cook_plot(fit)
```



# Binomial GLM - Plotting influence measures

I wrote the `dfbeta_plot()` function to easily plot the cook distances along with the identification of influential observations:

```
dfbeta_plot(fit)
```



# Residuals vs Fitted plot

The **residual vs fitted** plot is very useful to identify pattern in residuals and heteroskedasticity. Let's generate some data where the true model is  $y_i = \beta_0 + \beta_1 x + \beta_2 x^2$  and fit a model with and without the quadratic term<sup>7</sup>.

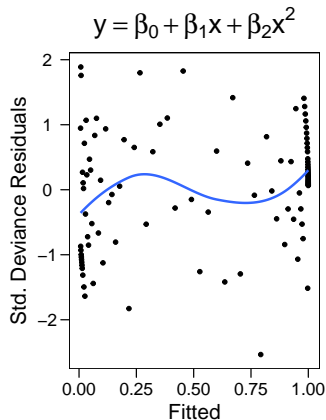
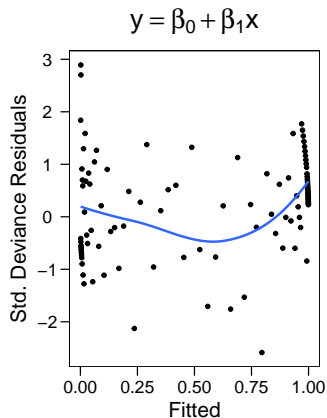
```
##      x y  n
## 1 0.00 1 50
## 2 0.01 1 50
## 3 0.02 1 50
## 4 0.03 0 50
## 5 0.04 0 50
## 6 0.05 0 50

fit_wrong <- glm(cbind(y, n - y) ~ x, data = dat, family = binomial(link = "logit"))
# poly 2 is a shortcut for fitting a quadratic term
fit_real <- glm(cbind(y, n - y) ~ poly(x, 2), data = dat, family = binomial(link = "logit"))
```

---

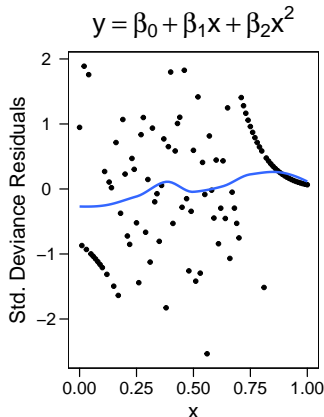
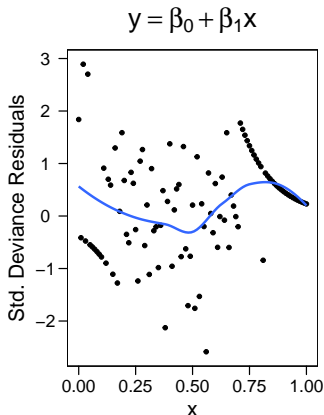
<sup>7</sup>Adapted from Dunn (2018, Figure 8.4)

# Residuals vs Fitted plot



# Residuals vs predictors

Another useful visual method is plotting residuals against each predictor  $x_j$ . In this case we have only one  $x$ :





# Binomial vs Binary

There are several practical differences between binomial and binary models:

- data structure
- fitting function in R
- residuals and residual deviance
- type of predictors #extra

# Binomial vs Binary data structure

The most basic Binomial regression is a vector of binary  $y$  values and a continuous or categorical predictor. Let's see a common data structure in this case:

```
n <- 30
x <- runif(n, 0, 1)
dat <- sim_design(ns = n, nx = list(x = x))
b0 <- qlogis(0.01)
b1 <- 8

dat |>
  sim_data(plogis(b0 + b1*x), "binomial") |>
  round(2) |>
  trim_df(4)
```

```
##   id    x    lp    y
## 1    1 0.8 0.86    1
## 2    2 0.83 0.88    1
## 3    3 0.34 0.14    0
## 4    4 0.26 0.07    0
## 5 ...   ...   ...   ...
## 6   27 0.3  0.1    0
## 7   28 0.3  0.1    0
## 8   29 0.88 0.92    1
## 9   30 0.12 0.02    0
```

# Binomial vs Binary data structure

Or equivalently with a categorical variable:

```
n <- 15
x <- c("a", "b")
dat <- sim_design(n, cx = list(x = x))
b0 <- qlogis(0.4)
b1 <- log(odds_ratio(0.7, 0.4))
dat$xd <- ifelse(dat$x == "a", 0, 1)

dat |>
  sim_data(plogis(b0 + b1*xd), "binomial") |>
  trim_df(4)
```

```
##      id  x x_c  xd lp  y
## 1     1  a   0   0 0.4  0
## 2     2  b   1   1 0.7  1
## 3     3  a   0   0 0.4  0
## 4     4  b   1   1 0.7  0
## 5 ... ..
## 6    27  a   0   0 0.4  0
## 7    28  b   1   1 0.7  0
## 8    29  a   0   0 0.4  0
## 9    30  b   1   1 0.7  0
```

# Binomial vs Binary data structure

When using a Binomial data structure we count the number of success for each level of  $x$ .  $nc$  is the number of 1 responses,  $nf$  is the number of 0 response out of  $nt$  trials:

```
x <- seq(0, 1, 0.05)
nt <- 10
nc <- rbinom(length(x), nt, plogis(qlogis(0.01) + 8*x))
dat <- data.frame(x, nc, nf = nt - nc, nt)

dat |>
  trim_df(4)
```

```
##      x  nc  nf  nt
## 1    0   0  10  10
## 2 0.05   0  10  10
## 3 0.1    0  10  10
## 4 0.15   0  10  10
## 5 ...   ...  ...  ...
## 6 0.85   9   1  10
## 7 0.9    9   1  10
## 8 0.95   9   1  10
## 9    1  10   0  10
```

# Binomial vs Binary data structure

With a categorical variable we have essentially a contingency table:

```
x <- c("a", "b")
nt <- 10
nc <- rbinom(length(x), nt, plogis(qlogis(0.4) + log(odds_ratio(0.7, 0.4))*ifelse(x == "a", 1, 0)))

datc <- data.frame(x, nc, nf = nt - nc, nt)
datc
```

```
##   x nc nf nt
## 1 a  9  1 10
## 2 b  4  6 10
```

# Binomial vs Binary: data structure

Clearly, expanding or aggregating data is the way to convert a binary into a binomial data structure and the opposite:

```
# from binomial to binary  
bin_to_binary(datc, nc, nt) |>  
  select(y, x) |>  
  trim_df()
```

```
##      y  x  
## 1    1  a  
## 2    1  a  
## 3    1  a  
## 4    1  a  
## 5 ...  ...  
## 6    0  b  
## 7    0  b  
## 8    0  b  
## 9    0  b
```

```
# from binary to binomial  
bin_to_binary(datc, nc, nt) |>  
  binary_to_bin(y, x)
```

```
## # A tibble: 2 x 4  
##   x      nc    nf    nt  
##   <chr> <dbl> <dbl> <int>  
## 1 a      9     1    10  
## 2 b      4     6    10
```

# Binomial vs Binary: - fitting function in R

```
# binary regression  
glm(y ~ x, family = binomial(link = "logit"))  
  
# binomial with cbind syntax, nc = number of 1s, nf = number of 0s, nc + nf = nt  
glm(cbind(nc, nf) ~ x, family = binomial(link = "logit"))  
  
# binomial with proportions and weights, equivalent to the cbind approach, nt is the total trials  
glm(nc/nt ~ x, weights = nt, binomial(link = "logit"))
```

# Binomial vs Binary: residuals and residual deviance

A more relevant difference is about the residual analysis. The binary regression has different residuals compared to the binomial model fitted on the same dataset<sup>8</sup>.

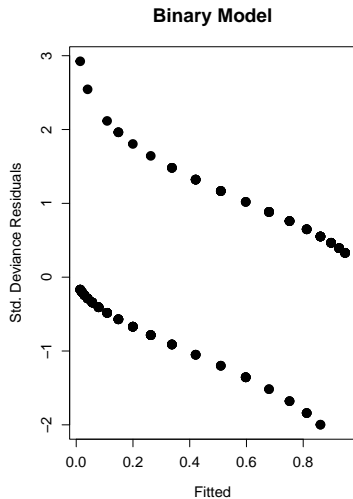
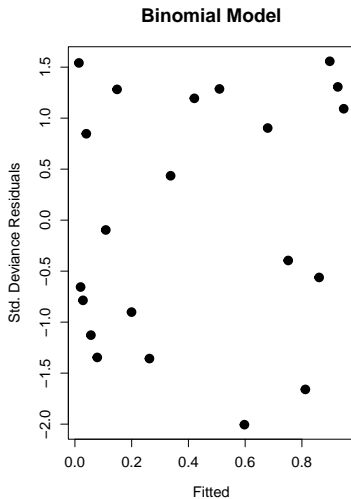
```
fit_binomial <- glm(nc/nt ~ x, weights = nt, data = dat_binomial, family = binomial(link = "logit"))
fit_binary <- glm(y ~ x, data = dat_binary, family = binomial(link = "logit"))
```

---

<sup>8</sup>To note, the **binned residuals** could be considered as an attempt to mitigate the binary residuals problem by creating bins of fitted/residual values similarly to fitting a binomial model.



# Binomial vs Binary: residuals and residual deviance



# Binomial vs Binary: residuals and residual deviance

The residual deviance is also different. In fact, there is more residual deviance on the binary compared to the binomial model. However, comparing two binary and binomial models actually leads to the same conclusion. In other terms the deviance seems to be on a different scale<sup>9</sup>:

```
fit0_binary <- update(fit_binary, . ~ -x) # null binary model
fit0_binomial <- update(fit_binomial, . ~ -x) # null binary model
```

```
anova(fit_binomial, fit0_binomial, test = "Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: nc/nt ~ x
## Model 2: nc/nt ~ 1
##      Resid. Df Resid. Dev Df Deviance    Pr(>Chi)
## 1          19      25.471
## 2          20     143.146 -1   -117.67 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
anova(fit_binary, fit0_binary, test = "Chisq")
```

```
## Analysis of Deviance Table
##
## Model 1: y ~ x
## Model 2: y ~ 1
##      Resid. Df Resid. Dev Df Deviance    Pr(>Chi)
## 1          208      167.25
## 2          209     284.92 -1   -117.67 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

---

<sup>9</sup>Thanks to Prof. Altoè for this suggestion

# Binomial vs Binary: type of predictors #extra

This point is less relevant in this course but important in general. Usually, **binary regression** is used when the predictor is at the *trial level* whereas **binomial regression** is used when the predictor is at the *participant level*. When both levels are of interests one should use a mixed-effects model where both levels can be modelled.

- the probability of correct responses during an exam as a function of the question difficulty
- the probability of passing the exam as a function of the high-school background

# Binomial vs Binary: type of predictors #extra

*The probability of correct responses during an exam as a function of the question difficulty*

- each question (i.e., *trial*) has a 0/1 response and a difficulty level
- we are modelling a single person

```
dat <- expand_grid(id = 1, question = 1:30)
dat$y <- rbinom(nrow(dat), 1, 0.7)
dat$difficulty <- sample(1:5, nrow(dat), replace = TRUE)

dat |>
  select(id, question, difficulty, y) |>
  trim_df()
```

##	id	question	difficulty	y
## 1	1	1	1	1
## 2	1	2	4	1
## 3	1	3	4	1
## 4	1	4	5	1
## 5	...	...	...	...
## 6	1	27	3	1
## 7	1	28	1	0
## 8	1	29	5	1
## 9	1	30	3	1

# Binomial vs Binary: type of predictors #extra

*the probability of passing the exam as a function of the high-school background*

- each “background” has different students that passed or not the exam (0/1)

```
## background nc nf nt
## 1      math 18 12 30
## 2 chemistry 15  5 20
## 3      art  6  4 10
## 4     sport 15  5 20
```

Or the binary version:

```
## y background nc nf nt
## 1 1      math 18 12 30
## 2 1      math 18 12 30
## 3 1      math 18 12 30
## 4 1      math 18 12 30
## 5 ...      ...  ...  ...
## 6 0     sport 15  5 20
## 7 0     sport 15  5 20
## 8 0     sport 15  5 20
## 9 0     sport 15  5 20
```

# Binomial vs Binary: type of predictors #extra

- To note that despite we can convert between the binary/binomial, the two models are not always the same. The *high-school background example* can be easily modelled either with a binary or binomial model because the predictor is at the participant level that coincides with the trial level.
- On the other side, the *question difficulty example* can only be modelled using a binary regression because each trial (0/1) has a different value for the predictor
- To include both predictors or to model multiple participants on the *question difficulty example* we need a mixed-effects (or multilevel) model where both levels together with the repeated-measures can be handled.

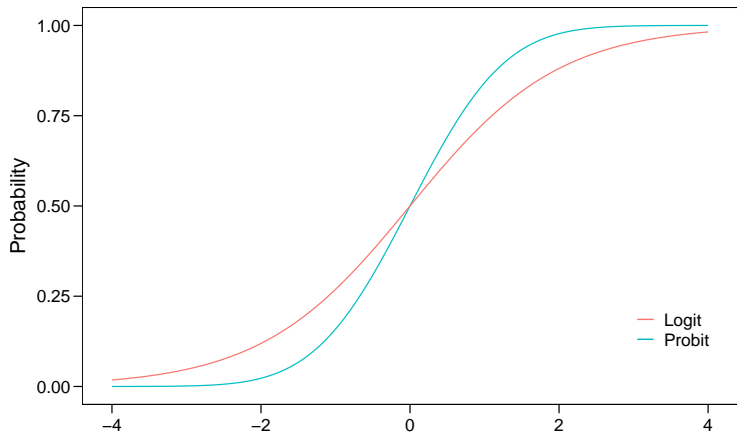
## Binomial GLM - Probit link

# Binomial GLM - Probit link

- The mostly used *link function* when using a binomial GLM is the **logit link**. The **probit** link is another *link function* that can be used. The overall approach is the same between **logit** and **probit** models. The only difference is the parameter interpretation (i.e., no odds ratios) and the specific link function (and the inverse) to use.
- The **probit** model use the **cumulative normal distribution** but the actual difference with a **logit** functions is negligible.



# Binomial GLM - Probit link



# Binomial GLM - Probit link

When using the **probit link** the parameters are interpreted as difference in *z-scores* associated with a unit increase in the predictors. In fact probabilities are mapped into *z-scores* using the cumulative normal distribution.

```
p1 <- 0.7  
p2 <- 0.5  
  
qlogis(c(p1, p2)) # log(odds(p1)), logit link
```

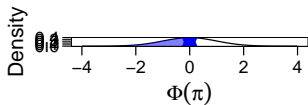
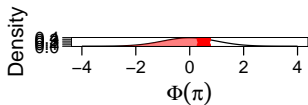
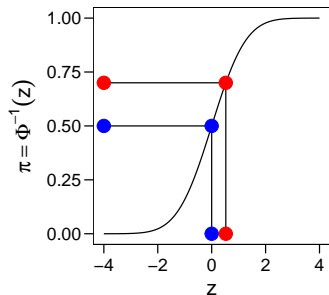
```
## [1] 0.8472979 0.0000000  
qnorm(c(p1, p2)) # probit link
```

```
## [1] 0.5244005 0.0000000  
log(odds_ratio(p1, p2)) # ~ beta1, logit link
```

```
## [1] 0.8472979  
pnorm(p1) - pnorm(p2) # ~beta1, probit link
```

```
## [1] 0.06657389
```

# Binomial GLM - Probit link



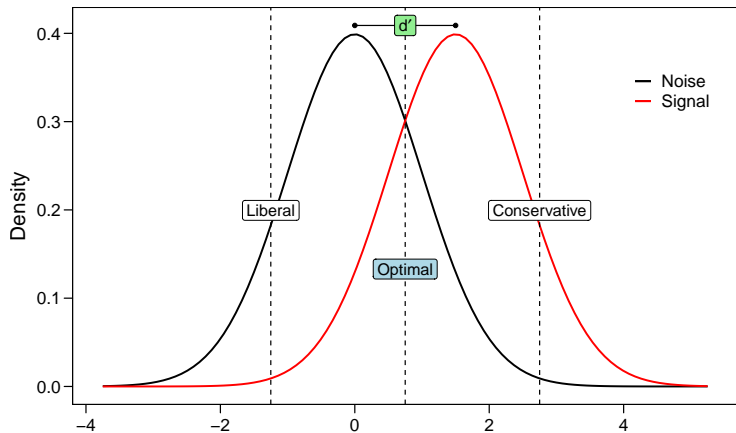
# Binomial GLM - Probit link and SDT #extra

One useful reason to use the **probit** link is when estimating ***Signal Detection*** parameters with a GLM approach [6].

The Signal Detection theory is a statistical approach to evaluate the ability of subject to discriminate between signal and noise.

In psychology, is used in cognitive-perceptual tasks and psychophysics to understand how well a participant is able to detect the presence of a stimulus or sound.

# Binomial GLM - Probit link and SDT #extra



# Binomial GLM - Probit link and SDT #extra

The two main parameters for a signal detection analysis are the  $d'$  and the **criterion**. The  $d'$  is the ability to discriminate between signal and noise (the distance between the two distributions of the previous slide) and the criterion is the tendency to say *yes* (liberal) or *no* (conservative) of the subject. The core of the SDT analysis is separately estimating the  $d'$  and the **criterion** thus the ability to discriminate regardless the actual decision strategy.

$$d' = \Phi(HIT) - \Phi(FA)$$

$$c = \frac{\Phi(HIT) + \Phi(FA)}{2}$$

Where **HIT** is the **sensitivity** defined for the ROC analysis (*say signal present when the signal is actually present*) and the **FA** (*false alarms*) is  $1 - \text{specificity}$  defined for the ROC analysis (*say signal present when the signal is actually absent*).

# Binomial GLM - Probit link and SDT #extra

The signal detection formalize the problem as a system (e.g., a person) that take a decision based on a noisy signal. Each decision, the evidence (i.e., the signal) is compared to an internal criterion. If the perceived signal exceed the criterion, the person say **yes** otherwise **no**.

- the signal distribution is  $Signal \sim \mathcal{N}(d', 1)$
- the noise distribution is  $Noise \sim \mathcal{N}(0, 1)$
- the optimal criterion is the midpoint between signal and noise  $\frac{d'}{2}$  but a person could choose a different criterion (e.g., *conservative* or *liberal*)

# Binomial GLM - Probit link and SDT #extra

Example: A group of clinicians need to guess the presence of learning disorders by observing a class of children during a day. Within the class 50% have learning disorders and 50% have no diagnosis.

- **HIT**: the clinician say *yes* and the children *has a disorder*
- **FA**: the clinician say *yes* and the children *do not has a disorder*

The  $d'$  is the overall ability of a clinician to classify the class and the **criterion** is the tendency to say *yes* or *no* regardless the reality. We can use some simulated data with  $n = 100$  children and we want to assess the ability of the clinician detecting the learning disorder:

##	say_disorder	has_disorder	x
## 1	1	1	0.5417636
## 2	1	1	2.3552751
## 3	1	1	1.5729267
## 4	1	1	1.1531070
## 5	1	1	1.5694414
## 6	1	1	1.1655376



# Binomial GLM - Probit link and SDT #extra

```
sdt$sdt <- dplyr::case_when(
  sdt$say_disorder == 1 & sdt$has_disorder == 1 ~ "hit",
  sdt$say_disorder == 0 & sdt$has_disorder == 1 ~ "miss",
  sdt$say_disorder == 1 & sdt$has_disorder == 0 ~ "fa",
  sdt$say_disorder == 0 & sdt$has_disorder == 0 ~ "cr"
)

sdt_tab <- table(sdt$sdt)
sdt_tab
```

```
##
##   cr   fa  hit miss
##  37   13   36   14

hr <- sdt_tab["hit"] / (sdt_tab["hit"] + sdt_tab["miss"])
far <- sdt_tab["fa"] / (sdt_tab["fa"] + sdt_tab["cr"])
```

# Binomial GLM - Probit link and SDT #extra

We could manually calculate the  $d'$  and criterion:

```
# dprime  
qnorm(hr) - qnorm(far)
```

```
##      hit  
## 1.226187
```

```
# criterion, negative = tendency to say yes  
-((qnorm(hr) + qnorm(far))/2)
```

```
##      hit  
## 0.03025195
```

# Binomial GLM - Probit link and SDT #extra

Or fitting a GLM with a **probit** link predicting the response by the reality:

```
fit <- glm(say_disorder ~ has_disorder,  
          # this is for having the same sign as the standard formula  
          contrasts = list(has_disorder = -contr.sum(2)/2),  
          data = sdt,  
          family = binomial(link = "probit"))
```

```
# the intercept is the criterion  
-coef(fit)[1] # flipping the sign is required
```

```
## (Intercept)  
## 0.03025195
```

```
# the slope is the dprime  
coef(fit)[2]
```

```
## has_disorder1  
## 1.226187
```

# References I

- [1] A. Gelman, J. Hill, and A. Vehtari, *Regression and other stories*. Cambridge University Press, 2020. doi: 10.1017/9781139161879.
- [2] A. Gelman and J. Hill, *Data analysis using regression and Multilevel/Hierarchical models*. Cambridge University Press, 2006. doi: 10.1017/CBO9780511790942.
- [3] D. McFadden, "Regression-based specification tests for the multinomial logit model," *J. Econom.*, vol. 34, no. 1, pp. 63–82, Jan. 1987, doi: 10.1016/0304-4076(87)90067-4.
- [4] T. Tjur, "Coefficients of determination in logistic regression Models—A new proposal: The coefficient of discrimination," *Am. Stat.*, vol. 63, no. 4, pp. 366–372, Nov. 2009, doi: 10.1198/tast.2009.08210.
- [5] P. K. Dunn and G. K. Smyth, "Randomized quantile residuals," *J. Comput. Graph. Stat.*, vol. 5, no. 3, pp. 236–244, Sep. 1996, doi: 10.1080/10618600.1996.10474708.
- [6] L. T. DeCarlo, "Signal detection theory and generalized linear models," *Psychol. Methods*, vol. 3, no. 2, pp. 186–205, Jun. 1998, doi: 10.1037/1082-989X.3.2.186.

[filippo.gambarota@unipd.it](mailto:filippo.gambarota@unipd.it)

[github.com/filippogambarota](https://github.com/filippogambarota)