Introduction to Generalized Linear Models

Filippo Gambarota

University of Padova

2022/2023

Outline

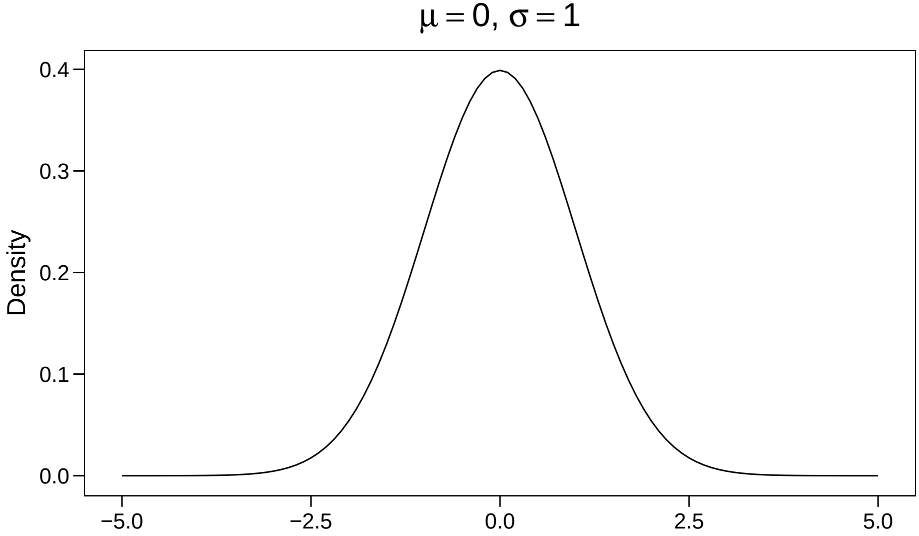
Beyond the Gaussian distribution

Quick recap about Gaussian distribution

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

Were μ is the **mean** and σ is the **standard deviation**

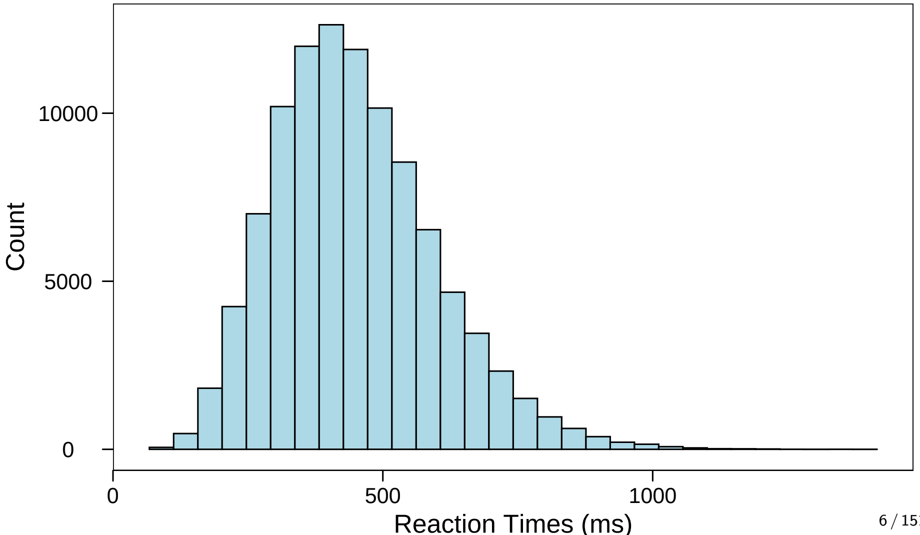
Quick recap about Gaussian distribution



But not always gaussian-like variables!

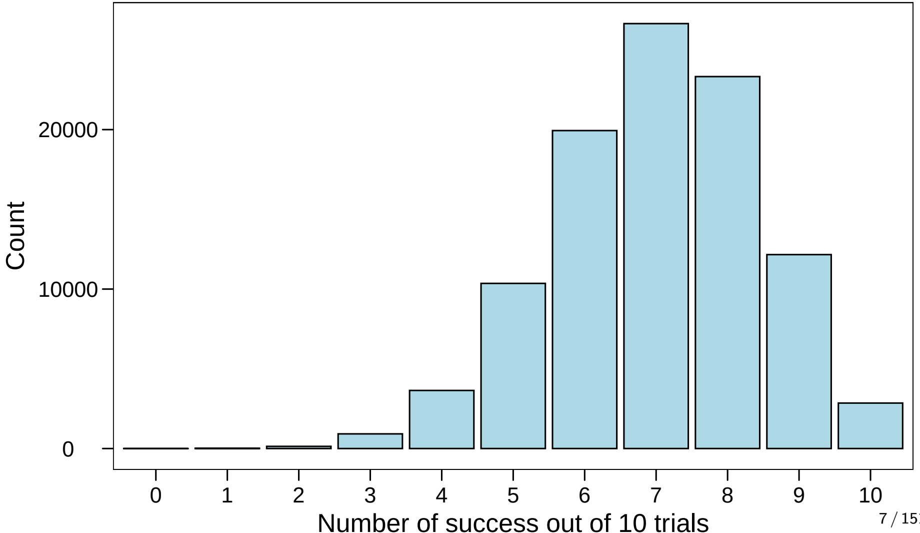
Reaction times

Measuring reaction times during a cognitive task. Non-negative and proably skewed data.

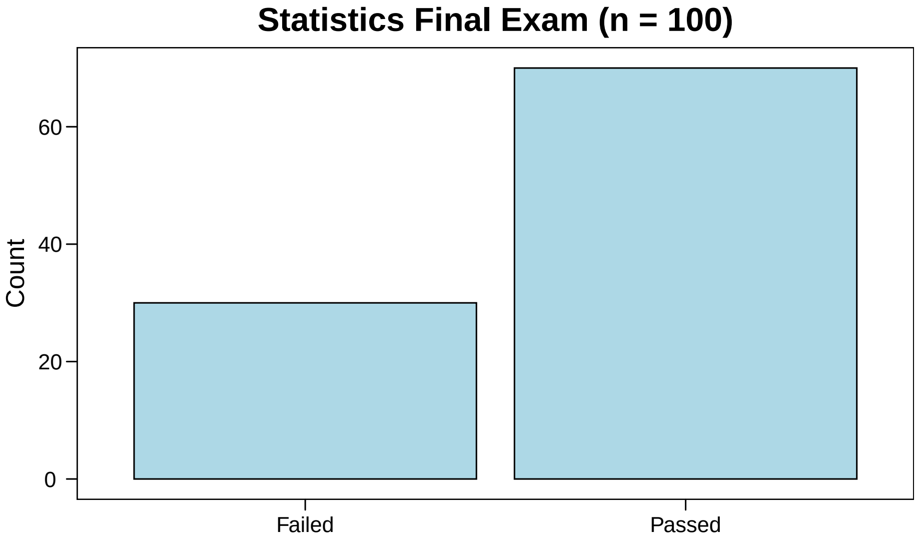


Binary outcomes

Counting the number of people passing the exam out of the total.
Discrete and non-negative. A series of binary (i.e., *bernoulli*) experiments.

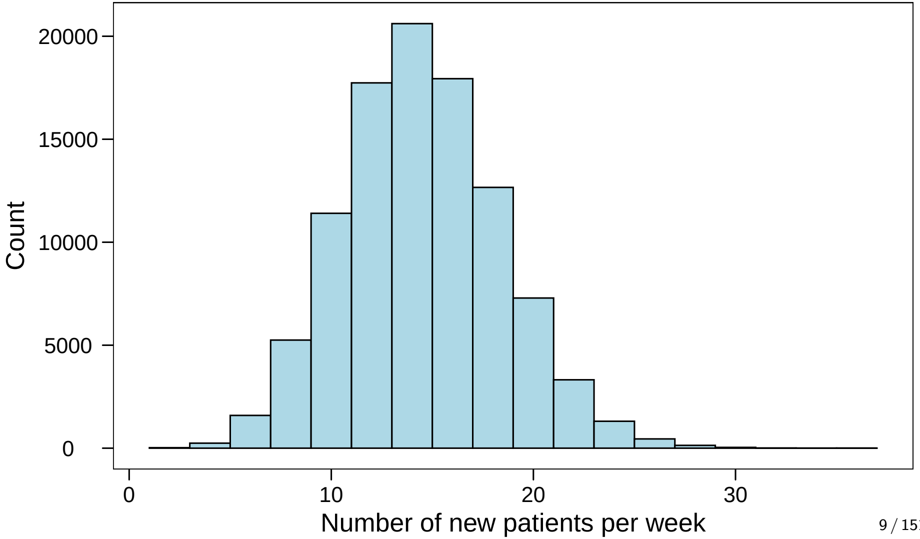


Binary outcomes

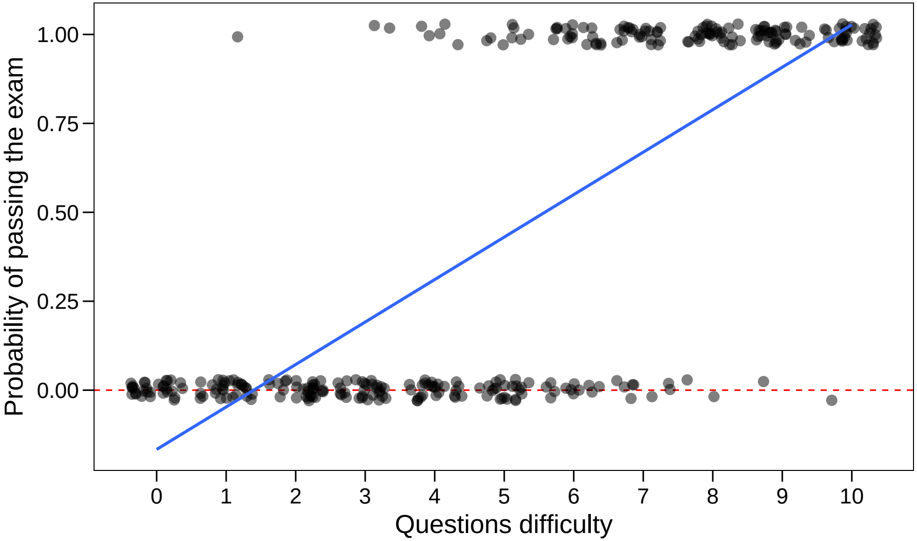


Counts

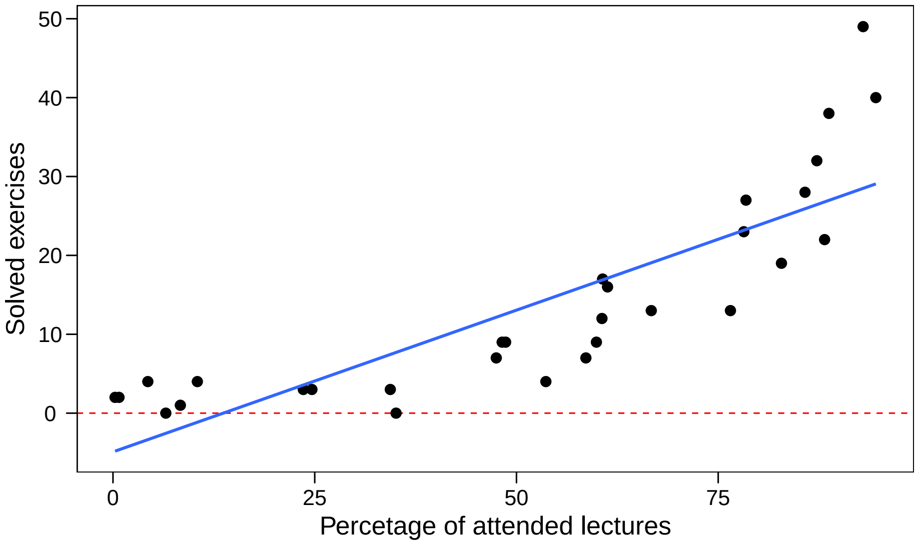
Counting the number of new patients per week. Discrete and non-negative values.



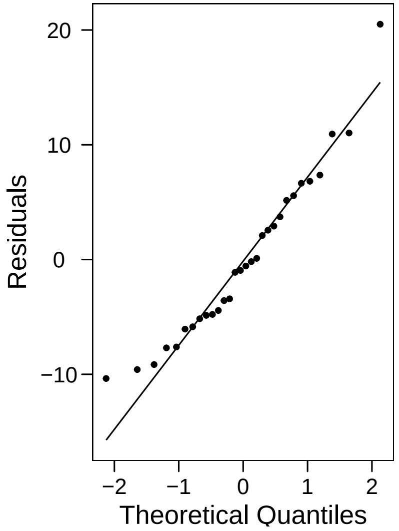
Should we use a linear model for these variables?

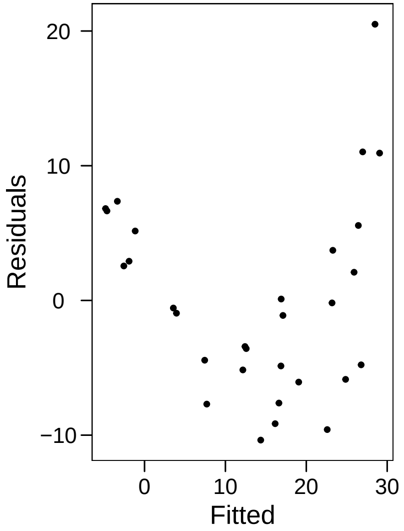


Should we use a linear model for these variables?



Should we use a linear model for these variables?





A new class of models

- We need that our model take into account the **features of our response variable**
- We need a model that, **with appropriate transformation**, keep **properties of standard linear models**
- We need a model that is **closer to the true data generation process**

Let's switch to Generalized Linear Models!

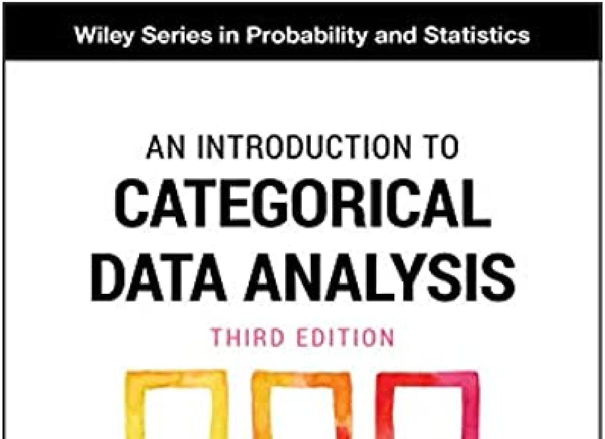
Generalized Linear Models

Main references

For a basic and well written introduction about GLM, especially the Binomial GLM

- Chapters: 3 (intro GLMs), 4-5 (Binomial Logistic Regression)

```r
knitr::include_graphics("img/agresti2019-intro-to-categorical.jpg")
```

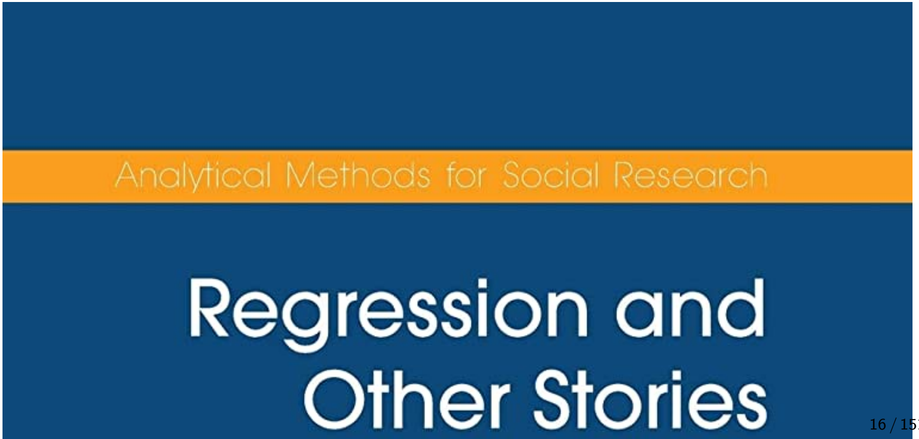


Main references

Great resource for interpreting Binomial GLM parameters:

- Chapters: 13-14 (Binomial Logistic GLM), 15 (Poisson and others GLMs)

```r
knitr::include_graphics("img/gelman2020-reg-and-other-stories.jpg")
```



General idea

- models that assume distributions other than the normal distributions
- models that considers non-linear relationships

Recipe for a GLM

- **Random Component**
- **Systematic Component**
- **Link Function**

Random Component

The **random component** of a GLM identify the response variable Y and the appropriate probability distribution. For example for a numerical and continous variable we could use a Normal distribution (i.e., a standard linear model). For a discrete variable representing counts of events we could use a Poisson distribution.

Systematic Component

The **systematic component** or *linear predictor* of a GLM is the combination of explanatory variables i.e. $\beta_0 + \beta_1 x_1 + ... + \beta_p x_p$.

Link Function

The **link function** $g(\mu)$ is the function that connects the expected value (i.e., the mean μ) of the probability distribution (i.e., the random component) with the *linear combination* of predictors $g(\mu) = \beta_0 + \beta_1 x_1 + ... + \beta_p x_p$

The simplest **link function** is the **identity link** where $g(\mu) = \mu$ and correspond to the classic linear model. In fact, the linear regression is just a GLM with a **Gaussian random component** and the **identity** link function.

There are multiple **random components** and **link functions** for example with a 0/1 binary variable the usual choice is using a **Binomial** random component and the **logit** link function.

Relevant distributions

Binomial distribution

The probability of having k success (e.g., 0, 1, 2, etc.) out of n trials with a probability of success p is:

$$f(n, k, p) = Pr(X = k) = \binom{n}{k} p^k (1 - p)^{n-k}$$

The np is the mean of the binomial distribution and $np(1 - p)$ is the variance.

Bernoulli distribution

The **binomial** distribution is just a repetition of k **Bernoulli** trials. A single Bernoulli trial is:

$$\begin{aligned} f(x,p) &= p^x(1-p)^{1-x} \\ x &\in \{0,1\} \end{aligned}$$

The mean is p and the variance is $p(1-p)$

Bernoulli and Binomial

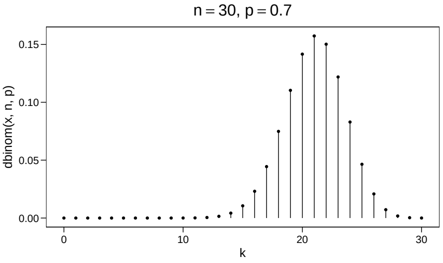
The simplest situation for a Bernoulli trials is a coin flip. In R:

```
n <- 1
p <- 0.7
rbinom(1, n, p) # a single bernoulli trial
```

```
## [1] 0
```

```
n <- 10
rbinom(1, 10, p) # n bernoulli trials
```

```
## [1] 7
```

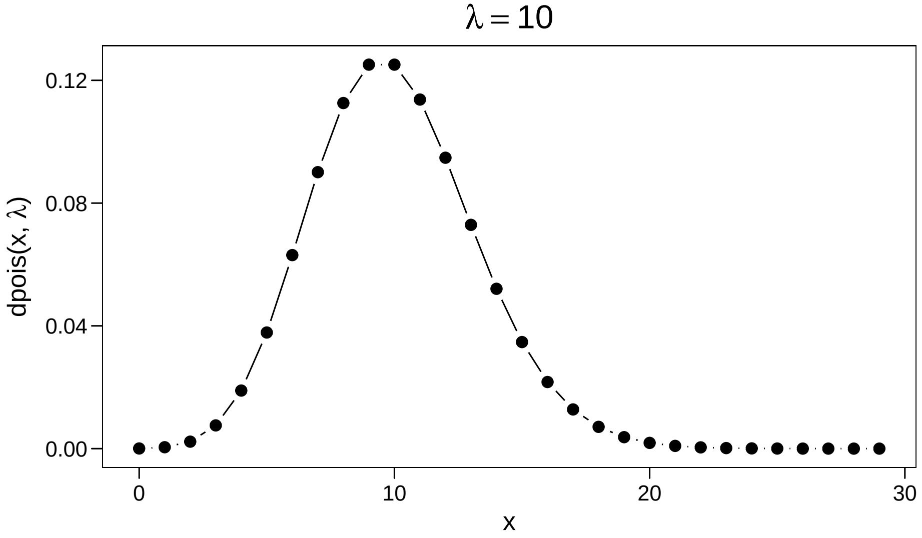


Poisson distribution

The number of events k during a fixed time interval (e.g., number of new user on a website in 1 week) is:

$$f(j, \lambda) = Pr(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

Poisson distribution



The mean and also the variance is λ.

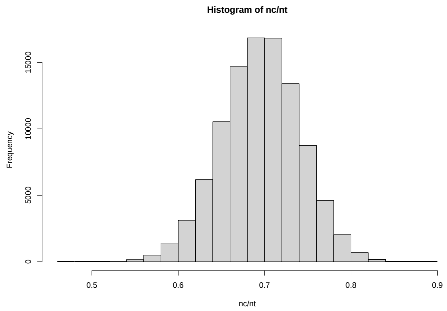
Data simulation *[EXTRA]*

Data simulation *[EXTRA]*

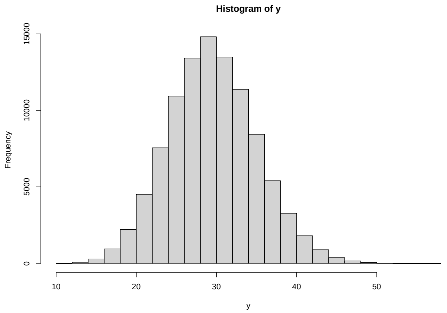
- During the course we will try to simulate some data. Simulating data is an amazing education tool to understand a statistical model.
- By simulating from a **generative model** we are doing **Monte Carlo Simulations** [1]

Data simulation *[EXTRA]*

```r
n <- 1e5 # number of experiments
nt <- 100 # number of subjects
p <- 0.7 # probability of success
nc <- rbinom(n, nt, p)
```



```r
n <- 1e5 # number of subjects
lambda <- 30 # mean/variance
y <- rpois(n, lambda)
```



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    0
## 3        1    0
## 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    41  25  66
##   1     9  25  34
##   Sum  50  50 100
```

Example: Passing the exam

Each cell probability π_{ij} is computed as π_{ij}/n

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

```
##      tv_shows
## exam     0    1  Sum
##   0   0.41 0.25 0.66
##   1   0.09 0.25 0.34
##   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). Firsly let's define *the odds of success* as:

$$odds = \frac{\pi}{1-\pi} \quad \pi = \frac{odds}{odds+1}$$

- 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:

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

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

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

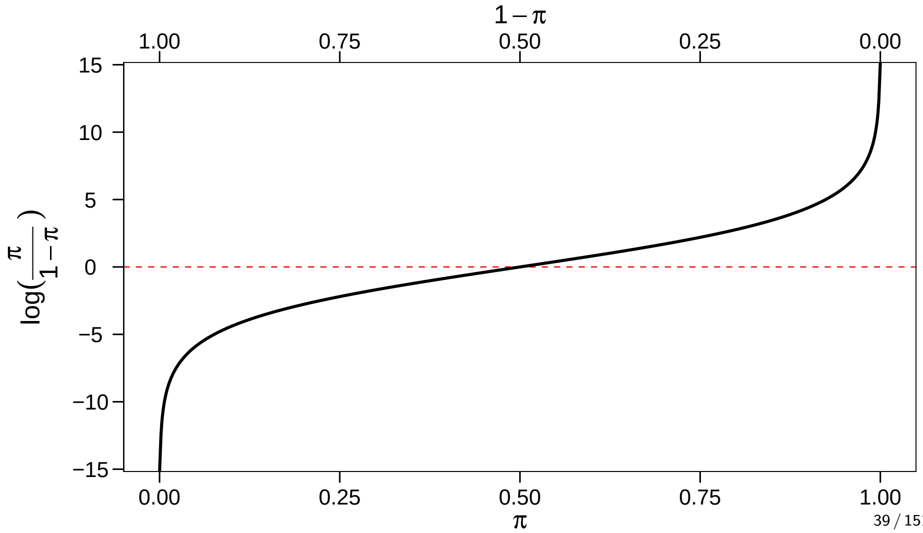
Example: Passing the exam - Odds Ratio

```r
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] 4.555556
```

Why using these measure?

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



Binomial GLM

Binomial GLM

- The **random component** of a Binomial GLM the binomial distribution with parameter π
- The **systematic component** is a linear combination of predictors and coefficients βX
- 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:

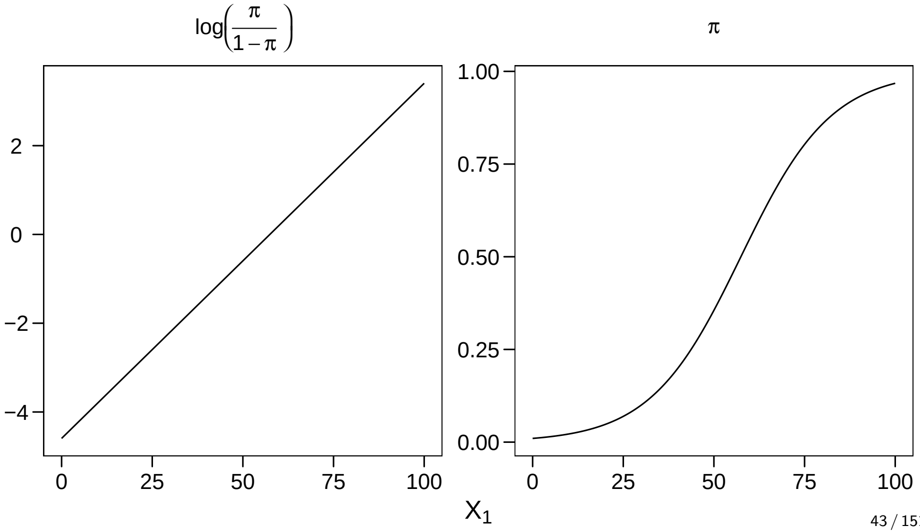
$$log\left(\frac{\pi}{1-\pi}\right) = \beta_0 + \beta_1 X_1 + ...\beta_p X_p$$

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

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

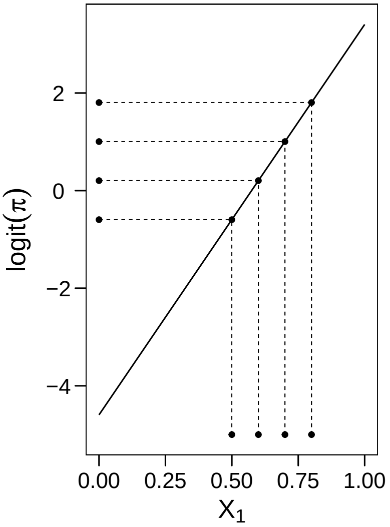
Binomial GLM - Logit Link

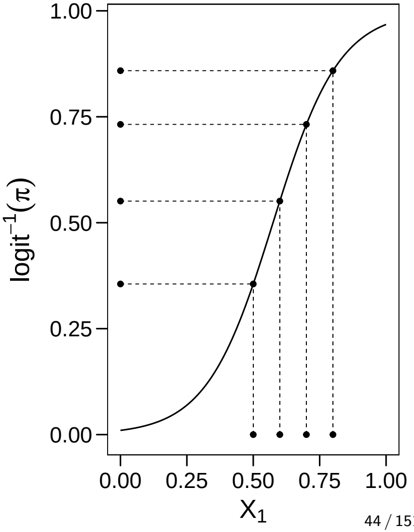
Thus with a single numerical predictor x the relationship between x and π in non-linear on the probability scale but linear on the logit scale.



Binomial GLM - Logit Link

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



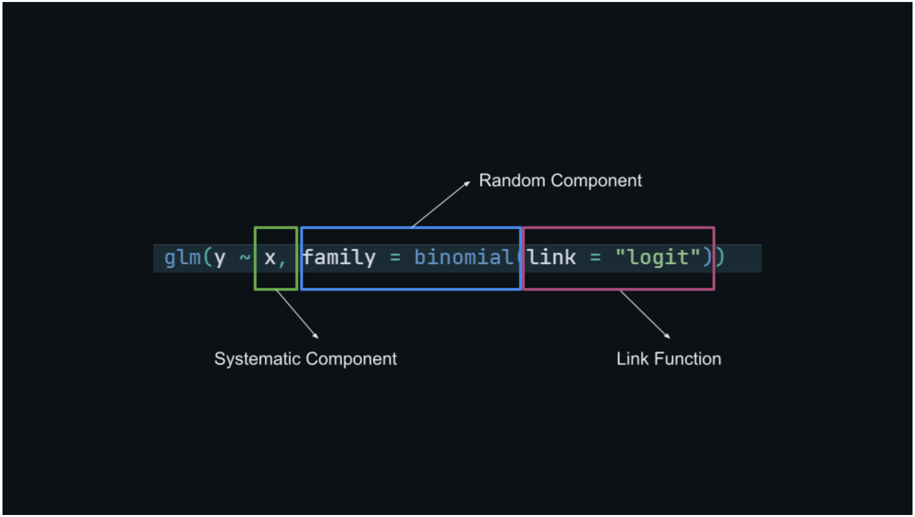


Binomial GLM - Model fitting

ciao

Binomial GLM - Model fitting in R

```r
knitr::include_graphics("img/glm-fitting-R.png")
```



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

Binomial GLM - Model fitting in R

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

```
dat |>
    trim_df()
```

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

```
# TODO add table here
```

Binomial GLM - Intercept only model

Let's start from the simplest model where the systematic part is only the intercept:

```
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
## -1.093  -1.093  -1.093   1.264   1.264
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.2007     0.2010  -0.998    0.318
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 137.63  on 99  degrees of freedom
## Residual deviance: 137.63  on 99  degrees of freedom
## AIC: 139.63
##
## Number of Fisher Scoring iterations: 3
```

Binomial GLM - Intercept only model

When fitting an intercept-only model, the parameter is the average value of the y variable. In our case, we are fitting a model $logis(\pi) = \beta_0$. Thus using the inverse of the link function we obtain the average y on the response scale:

In R the `logis` is the function to work with the logit transformation. `plogis()` is $logit^{-1}(\pi)$ and `qlogis` is $logit(\pi)$

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

```
## [1] 0.45
```

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

```
##     logit.(Intercept) inv-logit.(Intercept)
##            -0.2006707             0.4500000
```

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`:

```r
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
## -1.2462  -0.9448  -0.9448   1.1101   1.4294
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.5754     0.2946  -1.953   0.0508 .
## tv_shows      0.7357     0.4090   1.799   0.0721 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 137.63  on 99  degrees of freedom
## Residual deviance: 134.34  on 98  degrees of freedom
## AIC: 138.34
```

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
## 2.086957
```

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

```
## [1] 2.086957
```

Binomial GLM - Diagnostic

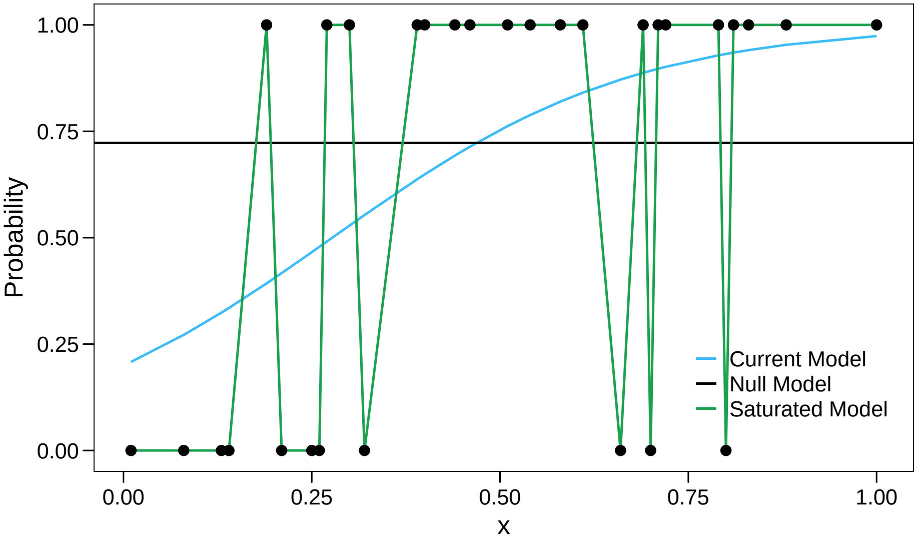
Binomial GLM - Residual Deviance

The residual deviance of a GLM is conceptually similar to the residual standard deviation of a standard linear model.

$$D = -2loglik(\beta)$$

The distance between a perfect model (i.e., saturated) and the current model can be considered as a goodness of fit measure. SPOILER the deviance is also useful to calculate a R^2 like measure comparing the null model with the actual model.

Binomial GLM - Residual Deviance[1]



[1] Adapted from https://bookdown.org/egarpor/SSS2-UC3M/logreg-deviance.html

Binomial GLM - Parameter Intepretation

Binomial GLM - Model Intepretation

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

- interpreting model coefficients on the linear and logit scale
- odds ratio (already introduced)
- the divide by 4 rule [2], [3]
- marginal effects
- predicted probabilities

Binomial GLM - Intepreting model coefficients

Models coeffiecients are intepreted 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 coeffiecients more intepretable or tailored to our research question.

Binomial GLM - Categorical predictors

We 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:

```
model.matrix(fit) |>
    as.data.frame() |>
    trim_df()
```

```
##   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 (β_0) is the reference level (default to 0 because is the first) and β_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 β_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:

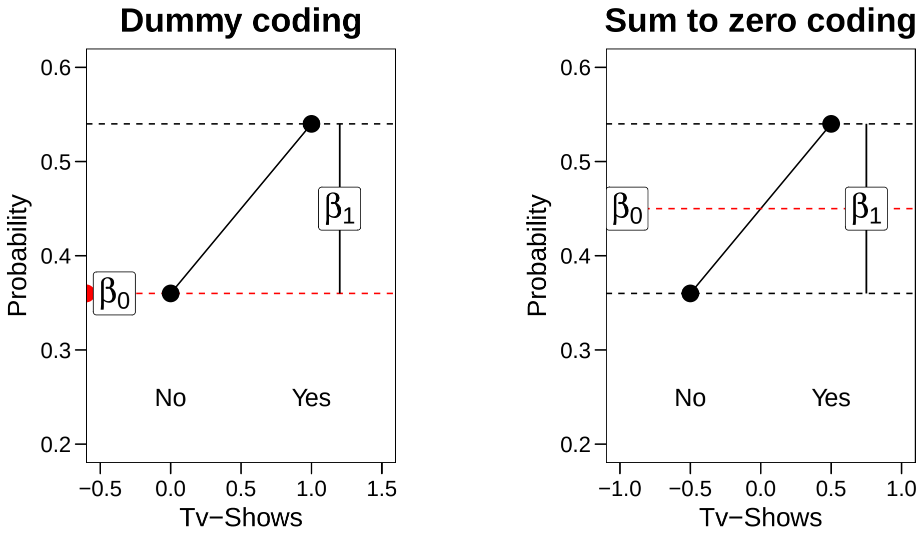
```r
dat$tv_shows0 <- ifelse(dat$tv_shows == 0, -0.5, 0.5)
fit <- glm(exam ~ tv_shows0, data = dat, family = binomial(link = "logit"))
# grand mean
mean(c(mean(dat$exam[dat$tv_shows == 1]), mean(dat$exam[dat$tv_shows == 0])))
```

```
## [1] 0.45
```

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

```
## (Intercept)
##   0.4483077
```

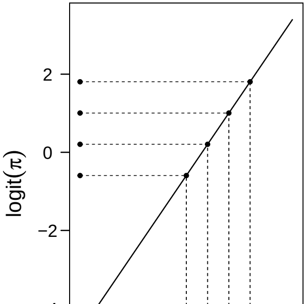
Binomial GLM - Categorical predictors

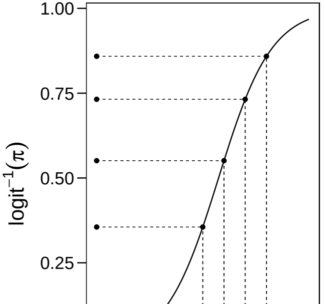


Binomial GLM - Numerical predictors

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 β_1 is interpreted in the same way as standard linear models only on the link-function scale. For the **binomial**

GLM the β_1 is the increase in the $log(odds(\pi))$ for a unit-increase in the x. In the response (probability) scale, the β_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 β values. The β_i of a centered predictor is the increase in y for a increase in one standard deviation of x.

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

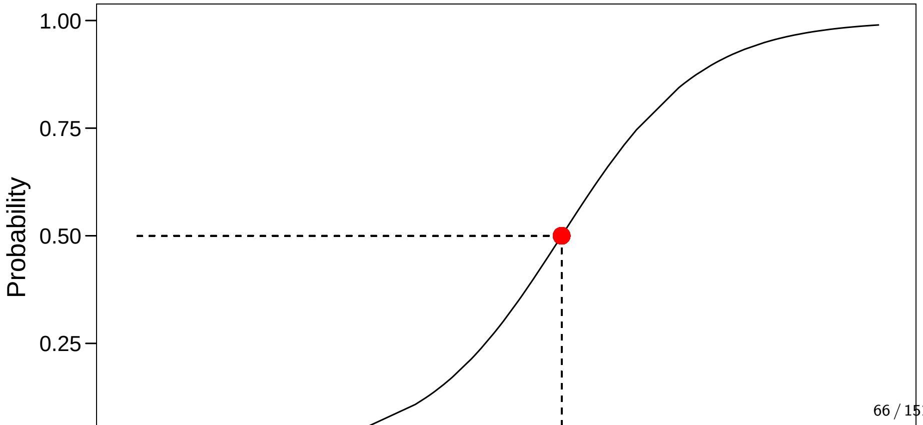
$$x_z = \frac{x_i - \hat{x}}{sigma_x}$$

Binomial GLM - Numerical predictors

```
x <- runif(100, 0, 1)
y <- rbinom(length(x), 1, plogis(qlogis(0.01) + x * 8))
dat <- data.frame(y, x)
dat$x0 <- scale(dat$x)
fit <- glm(y ~ x, data = dat, family = binomial(link = "logit"))
fit0 <- glm(y ~ x0, data = dat, family = binomial(link = "logit"))

raw <- dat |>
    add_predict(fit, type = "response") |>
    mutate(prl = qlogis(pr)) |>
    ggplot(aes(x = x, y = pr)) +
    geom_point(aes(x = x, y = y),
               position = position_jitter(height = 0.02)) +
    geom_line() +
    geom_point(x = 0, y = plogis(coef(fit)[1]),
               size = 5, col = "red") +
    geom_hline(yintercept = plogis(coef(fit)[1]),
               linetype = "dashed",
               col = "red") +
    mytheme() +
    xlab("x") +
    ylab("Probability") +
    annotate("label", x = 0, y = 0.20,
             label = latex2exp::TeX("$\\beta_0$"),
             size = 8) +
    ggtitle("Raw x")

centered <- dat |>
    add_predict(fit, type = "response") |>
    mutate(prl = qlogis(pr)) |>
    ggplot(aes(x = x0, y = pr)) +
    geom_point(aes(x = x0, y = y),
               position = position_jitter(height = 0.02)) +
    geom_line() +
```

Binomial GLM - Divide by 4 rule

The **divide by 4 rule** is a very easy but powerful way to rapidly evaluate the effect of a continous predictor on the probability. Given the non-linearity, the derivative of the logistic function (i.e., the slope) is maximal when predicts probabilities around ~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 - 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:

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

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

```
## (Intercept)
##   0.2781191
```

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
## 0.02910719 0.30722634
```

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

```
##         2
## 0.2781191
```

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 ~ -6.201 + 1.078*c(2.5, 5)
```

```
## [1] 0.02910719 0.30722634
```

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
# TODO add references
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] 0.1040218
```

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 0.1040 0.0028 37.5409 0.0000 0.0986 0.1095
```

Binomial GLM - Inference

Binomial GLM - Wald tests

The basic approach when doing inference with GLM is intepreting 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. We can calculate also the confidence interval for the model coefficients:

$$95\% CI = \beta_i \pm \Phi(\alpha/2)\sigma_{\beta_i}$$

Binomial GLM - Confidence intervals

Different types of confidence intervals

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

```
##                  2.5 %    97.5 %
## (Intercept) -9.0513352 -4.087549
## x            0.7223256  1.565710
```

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**:

```r
n <- 100
n_watching <- n/2 # just for simplicity
p_pass_yes <- 0.7
p_pass_no <- 0.3

dat <- data.frame(
    tv_shows = rep(c(1, 0), each = n_watching)
)

b0 <- 0.3 # P(passing|not watching)
b1 <- 3.5 # odds ratio between watching and not watching

# pn_from_or(0.3, 3.5) # P(passing|watching)

dat$exam <- rbinom(n, 1, plogis(qlogis(b0) + log(b1)*dat$tv_shows))
fit <- glm(exam ~ tv_shows, data = dat, family = binomial(link = "logit"))
```

```r
fits <- broom::tidy(fit)

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

# correct, wald-type confidence intervals
```

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:

```r
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  0.575  0.295              1 0.640 0.497 0.762
## 2        0 -0.754  0.303              1 0.320 0.204 0.463
```

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 x1_c x2_c        lp y
## 1  1  a  c  0.5  0.5 0.5505103 1
## 2  1  a  d  0.5 -0.5 0.2592193 0
## 3  1  b  c -0.5  0.5 0.3442153 0
## 4  1  b  d -0.5 -0.5 0.1304112 0
## 5  2  a  c  0.5  0.5 0.5505103 1
## 6  2  a  d  0.5 -0.5 0.2592193 1
```

We can fit the most complex model containing the two main effects and the interaction[2]:

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

[2] 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
## -1.2933  -0.8446  -0.3715   1.0658   2.3272
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.8379     0.2455  -3.413 0.000642 ***
## x11           1.0968     0.4909   2.234 0.025483 *
## x21           1.8105     0.4909   3.688 0.000226 ***
## x11:x21      -1.3900     0.9819  -1.416 0.156894
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 155.39  on 119  degrees of freedom
## Residual deviance: 133.86  on 116  degrees of freedom
## AIC: 141.86
##
## Number of Fisher Scoring iterations: 5
```

Then using `car::Anova()`. For each effect we have the χ^2 statistics and the associated p-value. The null hypothesis is that the specific factor did

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.

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

more details

Binomial GLM - Model comparison

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

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

anova(fit0, fit, test = "LRT")
```

```
## Analysis of Deviance Table
##
## Model 1: y ~ x1
## Model 2: y ~ x1 + x1:x2
##   Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
## 1       118     151.70
## 2       116     133.86  2   17.841 0.0001336 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The residual deviance and the p values are almost the same as using `car::Anova()`

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.

```
anova(fit1, fit2) # do not works properly
```

```
## Analysis of Deviance Table
##
## Model 1: y ~ x1
## Model 2: y ~ x2
##   Resid. Df Resid. Dev Df Deviance
## 1       118     151.70
## 2       118     140.28  0   11.417
```

```
anova(fit1, fit3) # same anova(fit2, fit3)
```

```
## Analysis of Deviance Table
##
```

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  2 149.8552
## fit3  3 150.4251
## fit1  2 161.2726
```

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

```
##      df      AIC
## fit3  3 142.0626
## fit2  2 144.2802
## fit1  2 155.6976
```

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 se:

- 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**McFadden1987-qq?**:

$$R^2 = 1 - \frac{log(L_m)}{log(L_0)}$$

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(fit2)
```

```
## # R2 for Generalized Linear Regression
##        R2: 0.097
##   adj. R2: 0.084
```

Nagelkerke's R^2

$$R^2 = \frac{1 - \frac{L_0}{L_m}^{2/n}}{1 - l_0^{2/n}}$$

This R^2 measure is actually an improvement of the R^2 by Cox and Snell **Cox1989-ql?** to correct the fact that in binomial models the range of the R^2 was not 0-1 as in standard R^2 measures **Nagelkerke1991-gr?**.

In R:

```
performance::r2_coxsnell(fit2) # uncorrected
```

```
## Cox & Snell's R2
##        0.1182893
```

```
performance::r2_nagelkerke(fit2) # uncorrected
```

```
## Nagelkerke's R2
##       0.1629165
```

Tjur's R^2

This measure is the easiest to interpret and calculate but can only be applied for binomial binary models **Tjur2009-ml?**. Is the absolute value of the difference between the proportions of correctly classifing $y = 1$ and $y = 0$ from the model:

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

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()`)

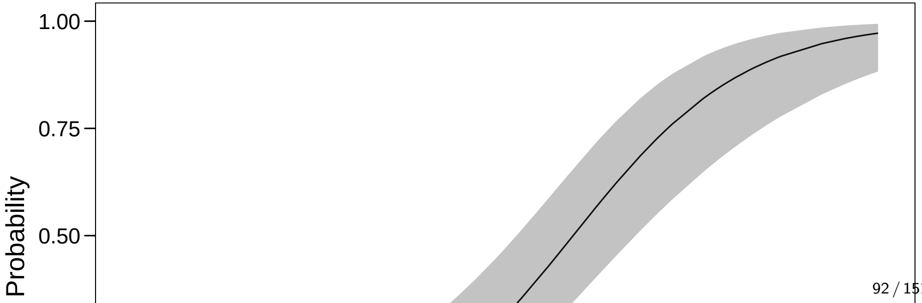
Binomial GLM - 1 categorical predictor

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

```r
n <- 100
n_watching <- n/2 # just for simplicity
p_pass_yes <- 0.7
p_pass_no <- 0.3

dat <- data.frame(
    tv_shows = rep(c(1, 0), each = n_watching)
)

b0 <- 0.3 # P(passing|not watching)
b1 <- 3.5 # odds ratio between watching and not watching

# pn_from_or(0.3, 3.5) # P(passing|watching)

dat$exam <- rbinom(n, 1, plogis(qlogis(b0) + log(b1)*dat$tv_shows))
dat$tv_shows <- factor(dat$tv_shows)

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

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

fits |>
    mutate(tv_shows = ifelse(tv_shows == 0, "No", "Yes"),
           lower = plogis(fit - se.fit * 2),
           upper = plogis(fit + se.fit * 2)) |>
    ggplot(aes(x = tv_shows, y = plogis(fit), ymin = lower, ymax = upper)) +
```

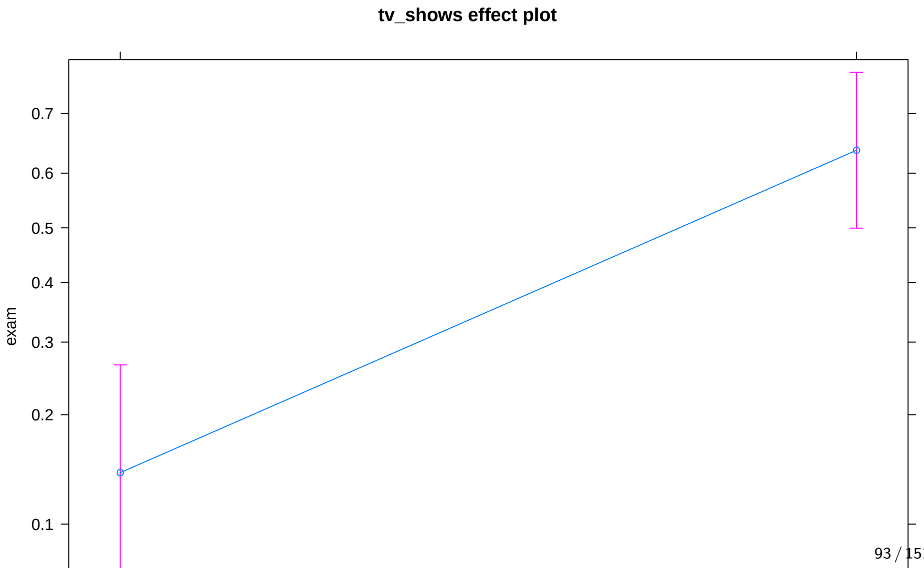
Binomial GLM - 1 numerical predictor

```r
x <- runif(100, 0, 1)
y <- rbinom(length(x), 1, plogis(qlogis(0.01) + x * 8))
dat2 <- data.frame(y, x)

fit2 <- glm(y ~ x, data = dat, family = binomial(link = "logit"))

dat2 |>
    add_predict(fit2, se.fit = TRUE) |>
    mutate(p = plogis(fit),
           lower = plogis(fit - 2 * se.fit),
           upper = plogis(fit + 2 * se.fit)) |>
    ggplot(aes(x = x, y = p, ymin = lower, ymax = upper)) +
    geom_line() +
    geom_ribbon(alpha = 0.3) +
    mytheme() +
    ylab("Probability")
```



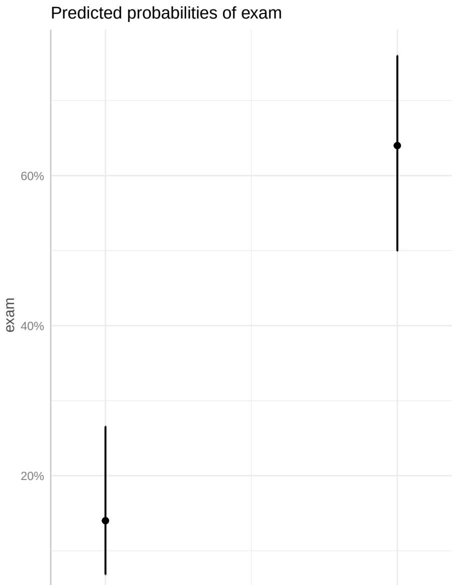
Binomial GLM - `allEffects()`

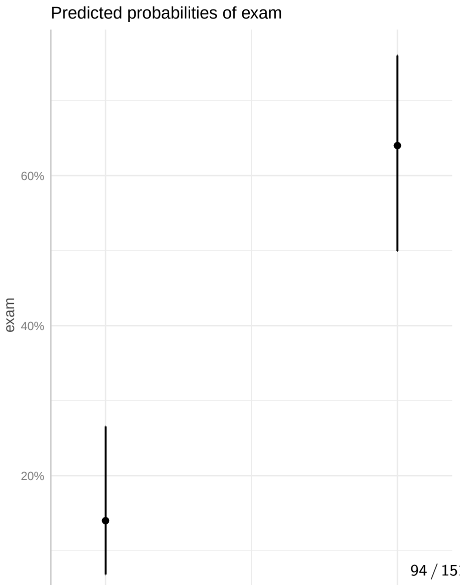
```
par(mfrow = c(1,2))
plot(effects::allEffects(fit))
```



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

```
plot(ggeffects::ggeffect(fit)[[1]]) + plot(ggeffects::ggeffect(fit)[[1]])
```

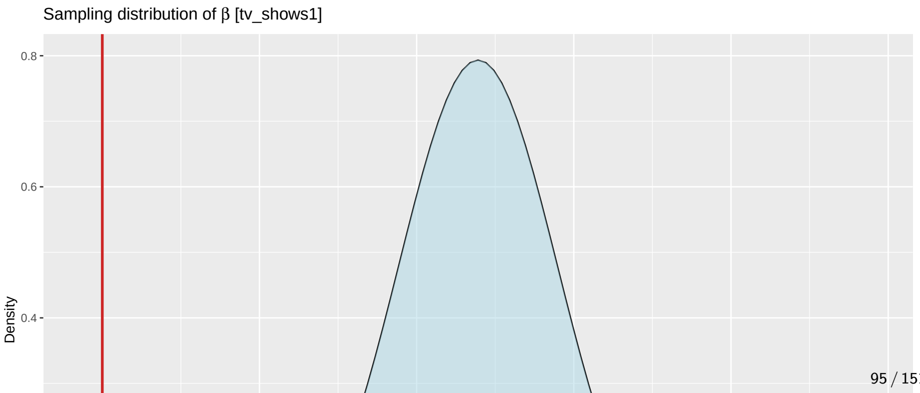




Binomial GLM - Plotting coefficients

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:

```
plot_param(fit, "tv_shows1")
```



Binomial GLM - Diagnostic

Binomial 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:

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

Binomial GLM - Residuals

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

- raw residuals
- pearson residuals
- standardized pearson residuals
- studentized residuals
- deviance residuals

Binomial GLM - Raw Residuals

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

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

In R:

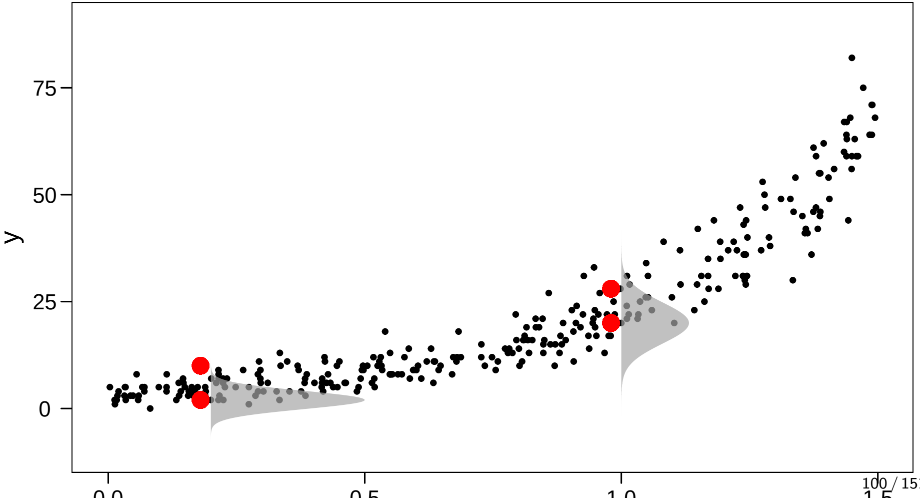
```r
# equivalent to residuals(fit, type = "response")
ri <- dat$y - fitted(fit)
ri[1:5]
```

```
## [1] NA NA NA NA NA
```

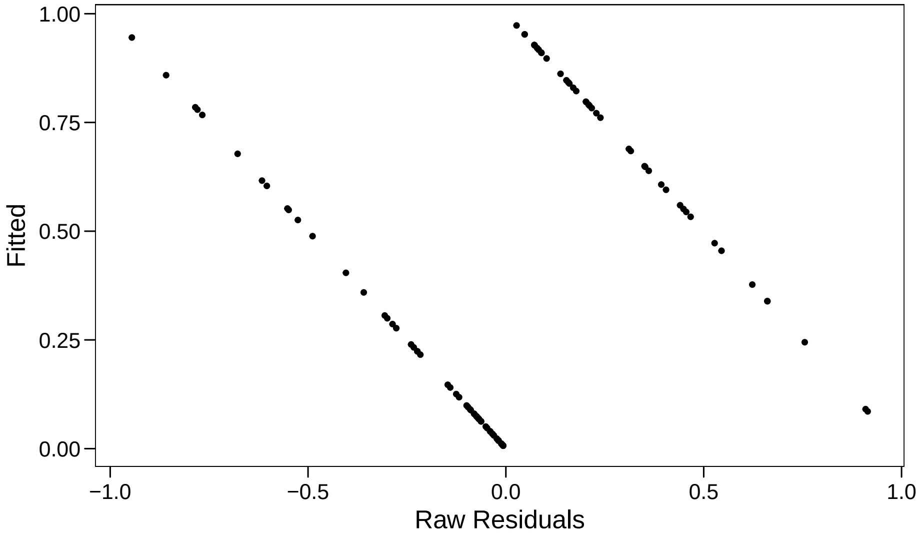
The problem is that in GLMs the mean and the variance of the distribution are usually correlated, creating a problem in residual analysis.

Binomial GLM - Raw Residuals

This plot[3] 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:



Binomial GLM - Raw Residuals



Binomial GLM - Binned (raw) Residuals

Gelman and colleagues [3] proposed a type of residuals called **binned residuals**. The idea is to divide values into a certain number of bins, calculate the average fitted/residual and plot the results. To calculate binned residuals we need to:

- 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 $SE = \frac{\hat{p}_j(1-p_j)}{n_j}$ where 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 ~95% of binned residuals should be within the CI if the model is true

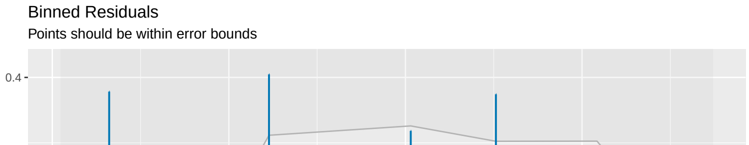
Binomial GLM - Binned (raw) Residuals

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 0.01168698 -0.01168698 10 0.006668362 0.01947290 0.003254965 0.001660727
## 2 0.04578606 -0.04578606 10 0.023128393 0.06771086 0.009370888 0.004781153
## 3 0.08052181  0.11947819 10 0.068913604 0.09090575 0.258839338 0.132063314
## 4 0.16224103 -0.16224103 10 0.095606409 0.23277749 0.034478936 0.017591617
## 5 0.30680345  0.09319655 10 0.239495747 0.37720297 0.311532031 0.158947834
## 6 0.50751842 -0.00751842 10 0.404238570 0.55209081 0.325792551 0.166223744
##        CI_low      CI_high group
## 1 -0.01494195 -0.008432018    no
## 2 -0.05515695 -0.036415173    no
## 3 -0.13936115  0.378317524   yes
## 4 -0.19671997 -0.127762096    no
## 5 -0.21833549  0.404728577   yes
## 6 -0.33331097  0.318274132   yes
```

```
plot(bres)
```



Binomial GLM - Pearson residuals

Pearson residuals are raw residuals divided by the standard deviation of each residual. The idea is to take into account the mean-variance relationship of GLMs. When model assumptions are respected, Pearson residuals have an approximate normal distribution: For the binomial of the binomial GLM:

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

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

```
##          1          2          3          4          5
## -0.2938928 -0.1076111  0.8870799  0.4003953 -1.4507864
```

Binomial GLM - Standardized Pearson residuals

The standardized pearson residuals are pearson residuals standardized using the hat value of each residual. The hat values are the diagonal of the hat matrix.

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

```r
# equivalent to residuals(fit, type = "pearson")
h <- hatvalues(fit) # diagonal of the hat matrix
ri_pearson_std <- ri_pearson / sqrt(1 - h)
ri_pearson_std[1:5]
```

```
##          1          2          3          4          5
## -0.2971215 -0.1081066  0.8957975  0.4053341 -1.4665296
```

Binomial GLM - Deviance residuals

The **deviance residuals** are calculated using the likelihood of the model and as for standard linear model the sum of the squared deviance residuals is the residual deviance used to assess the model goodness of fit.

$$sign(y_i - \hat{y}_i)\sqrt{-2(y_i log(\hat{p}_i) + (1 - y_i) log(1 - \hat{p}_i)}$$

```r
# equivalent to residuals(fit, type = "deviance")
ri_deviance <- sign(ri)*sqrt(-2*(dat$y*log(pi) + (1 - dat$y)*log(1 - pi)))
ri_deviance[1:5]
```

```
##          1          2          3          4          5 
## -0.4070495 -0.1517472  1.0774863  0.5453305 -1.5052862
```

```r
# residual deviance
sum(ri_deviance^2)
```

```
## [1] 82.10457
```

```r
fit$deviance
```

```
## [1] 82.10457
```

Also the **deviance residuals** tends to have a normal distribution if the model assumptions are respected.

Binomial GLM - Quantile residuals

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.

```r
dat <- data.frame(x = 0:30)
dat$y <- pbinom(dat$x, 30, 0.5)
dat$yl <- pbinom(dat$x-1, 30, 0.5)

u <- 0.35

qres1 <- dat |>
    ggplot(aes(x = x, y = y)) +
    geom_point(shape = 19, size = 4) +
    geom_point(aes(x = x, y = yl), shape = 21,size = 4) +
    geom_segment(aes(x = x, xend = x+1, y = y, yend = y)) +
    geom_segment(x = 0, xend = 25, y = pbinom(13, 30, 0.5), yend = pbinom(13, 30, 0.5),
                 linewidth = 0.5) +
    geom_segment(x = 0, xend = 25, y = pbinom(14, 30, 0.5), yend = pbinom(14, 30, 0.5),
                 linewidth = 0.5) +
    geom_label(x = 25, y = pbinom(13, 30, 0.5), label = sprintf("a = %.2f", pbinom(13, 30, 0.5))) +
    geom_label(x = 25, y = pbinom(14, 30, 0.5), label = sprintf("b = %.2f", pbinom(14, 30, 0.5))) +
    geom_segment(x = 0, xend = 10, y = u, yend = u) +
    geom_segment(x = 14, xend = 14, y = 0, yend = pbinom(14, 30, 0.5)) +
    geom_label(x = 10, y = u, label = sprintf("u = runif(1, a, b) = %.2f", u)) +
    geom_point(x = 0, y = u, size = 3, col = "red") +
    geom_text(x = 15, y = 0, label = "y = 14") +
    xlab("y") +
    ylab("CDF") +
    mytheme()

qres2 <- ggplot() +
```

Binomial GLM - Quantile residuals

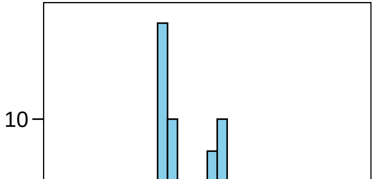
Quantile residuals are very useful especially for Discrete GLMs (binomial and poisson) and are exactly normally distributed (under respected model assumptions) compared to **deviance** and **pearson** residuals **Dunn1996-yd?**. They can be calculated using the `statmod::qresid(fit)` function. Authors suggest to run the function 4 times to disentangle between the randomization and the systematic component.

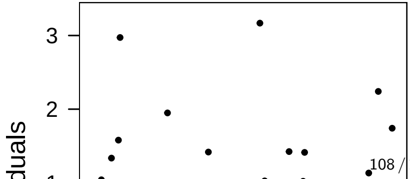
```
statmod::qresid(fit)[1:5]
```

```
## [1]  1.29722567  0.35413849 -0.06133714  2.66169931 -3.15204257
```

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

```
## [1] -1.11032250  2.13578212  0.07372524  1.29436061 -0.68619078
```





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 $logit^{-1}(\hat{y}_i) > 0.5$ and guessing $y_i = 0$ if $logit^{-1}(\hat{y}_i) > 0.5$ 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:

```
## ```r
##
## error_rate <- function(fit){
##     pi <- predict(fit, type = "response")
##     yi <- fit$y
##     cr <- mean((pi > 0.5 & yi == 1) | (pi < 0.5 & yi == 0))
##     1 - cr # error rate
## }
##
## ```
```

Binomial GLM - Classification accuracy/Error rate

```r
error_rate(fit)
```

```
## [1] 0.19
```

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):

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

```
## [1] 0.43
```

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

The **error rate** of the null model is 1 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):

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

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

```
##    yi
## pi  0.00666836211414141 0.0068836433178804 0.00699996377966199
##   0                   1                  1                   1
##   1                   0                  0                   0
##    yi
## pi  0.00788398828411116 0.00895048985557412 0.0114475821244671
##   0                   1                   1                  1
##   1                   0                   0                  0
##    yi
## pi  0.0114481360735971 0.0178851953680463 0.0192295694124393 0.0194728983324764
##   0                  1                  1                  1                  1
##   1                  0                  0                  0                  0
##    yi
## pi  0.023128392691948 0.0303171582383519 0.0339598686060344 0.0390471850614902
##   0                 1                  1                  1                  1
##   1                 0                  0                  0                  0
##    yi
## pi  0.0399725463590126 0.0474238268606509 0.0507409905603356 0.0625428180361926
##   0                  1                  1                  1                  1
##   1                  0                  0                  0                  0
```

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 - 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:

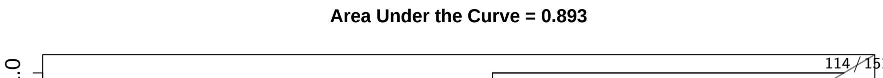
```
classify(fit, 0.5)
```

```
## $tp
## [1] 35
##
## $tn
## [1] 46
##
## $fp
## [1] 11
##
## $fn
## [1] 8
##
## $tpr
## [1] 0.8139535
##
## $tnr
## [1] 0.8070175
```

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:

```r
th <- c(-Inf, seq(0, 1, 0.001), Inf)
roc <- lapply(th, function(t) classify(fit, t))
roc <- bind_rows(roc)
roc$th <- th
auc <- pROC::auc(y ~ x, data = dat)

plot(roc$fpr, roc$tpr,
     type = "l",
     xlab = "1 - Specificity",
     ylab = "Sensitivity",
     cex.lab = 1.3,
     cex.axis = 1.3,
     lwd = 1.4,
     main = sprintf("Area Under the Curve = %.3f", auc))

abline(coef = c(0, 1),
       lwd = 1.4,
       col = alpha(rgb(0,0,0), 0.5))
```



Binomial GLM - Outliers and influential observations

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:

- The hat values
- Cook Distances
- DFBETAs

Quick recap about hatvalues in linear regression

The **hat matrix** H is calculated as $H = X\left(X^{\top}X\right)^{-1}X^{\top}$ is a $n \times n$ matrix where n is the number of observations. The diagonal of the H matrix containes the hatvalues 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 hatvalue

Quick recap about hatvalues in linear regression

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

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

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 
## 0.021614553 0.009146582 0.019368646 0.024220819 0.021354786 0.022288244 
##           7           8           9          10 
## 0.017149129 0.023434416 0.024202296 0.023593077
```

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

Binomial GLM - Standardized/Studentized residuals

The **response**, **pearson** and **deviance** residuals are defined as *raw* residuals. Using hatvalues is possible to compute the standardized version of these residuals. The residuals have different variance according to the actual value. Including the hatvalues stabilize the variance.

In R the standardize version of residuals can be obtained using:

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

```
##          1          2          3          4          5 
## -0.4115212 -0.1524460  1.0880751  0.5520572 -1.5216208 
```

```
rstudent(fit)[1:5] # studentized
```

```
##          1          2          3          4          5 
## -0.4093866 -0.1520990  1.0846746  0.5489670 -1.5204652 
```

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

```
##          1          2          3          4          5 
## -0.2971215 -0.1081066  0.8957975  0.4053341 -1.4665296 
```

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). ϕ 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[4].

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

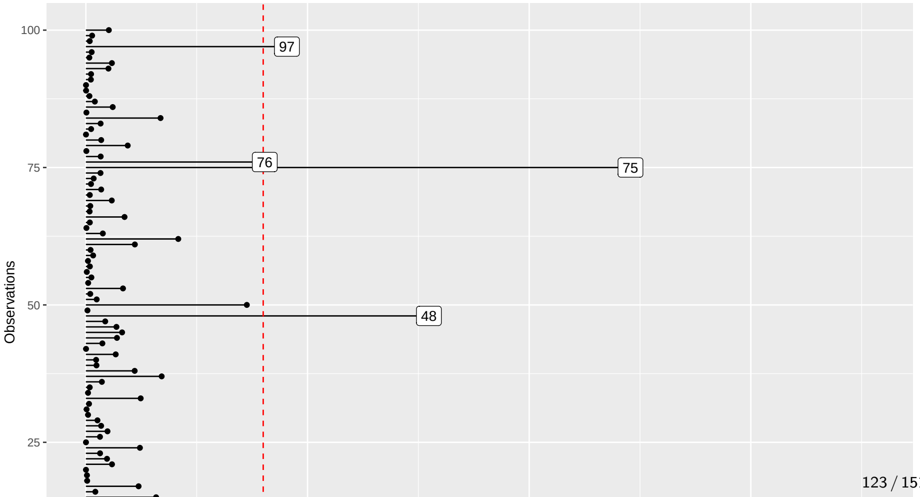
```
##        dfb.1_       dfb.x       dffit     cov.r       cook.d         hat
## 1 -0.06439696  0.05734211 -0.06655196 1.0389755 9.751565e-04 0.021614553
## 2 -0.01591268  0.01515424 -0.01592225 1.0295641 5.394163e-05 0.009146582
## 3  0.01629462  0.02992393  0.16742170 1.0110837 7.924707e-03 0.019368646
## 4 -0.05622325  0.07493470  0.09471362 1.0383099 2.039077e-03 0.024220819
## 5  0.05252065 -0.11679324 -0.24783070 0.9850027 2.346506e-02 0.021354786
## 6  0.35994799 -0.31758953  0.37483444 0.9229768 1.165848e-01 0.022288244
```

[4]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:

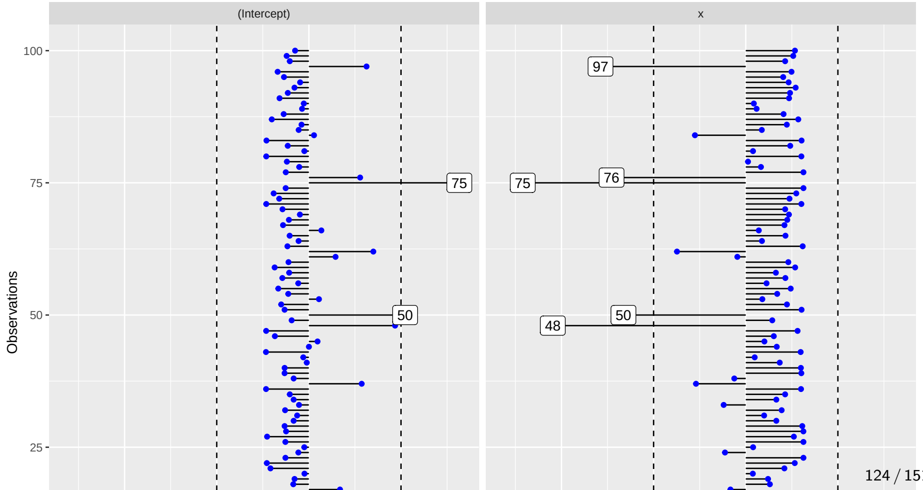
```
cook_plot(fit)
```



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)
```



Checking model assumptions

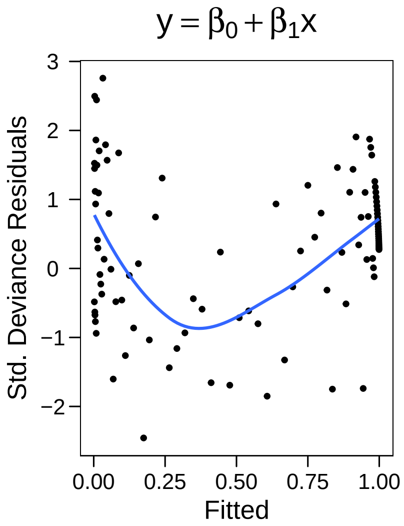
Residuals vs Fitted plot

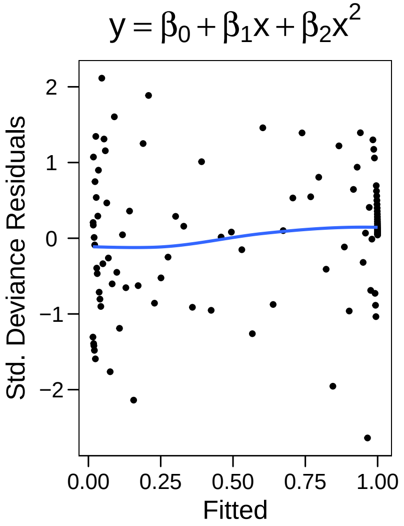
The **residual vs fitted** plot is very useful to identify pattern in residuals and heteroskedasticity. Let's generate some data where the true model is $y_i = \beta_0 + \beta_1 x + \beta_2 x^2$ and fit a model with and without the quadratic term[5].

```r
n <- 50
x <- seq(0, 1, 0.01)

dat <- data.frame(x)
dat$y <- rbinom(nrow(dat), n, plogis(qlogis(0.01) + 5*dat$x + 10*(dat$x)^2))
dat$n <- n
fit <- glm(cbind(y, n - y) ~ x, data = dat, family = binomial(link = "logit"))
fit2 <- glm(cbind(y, n - y) ~ poly(x, 2), data = dat, family = binomial(link = "logit"))

res <- data.frame(
    x, rd = rstandard(fit, type = "deviance"), rq = statmod::qresid(fit),
    rd2 = rstandard(fit2, type = "deviance"), rq2 = statmod::qresid(fit2),
    yi = fitted(fit), yi2 = fitted(fit2)
)

head(dat) # the real relationship is y ~ x + x^2
```

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

Residuals vs Fitted plot

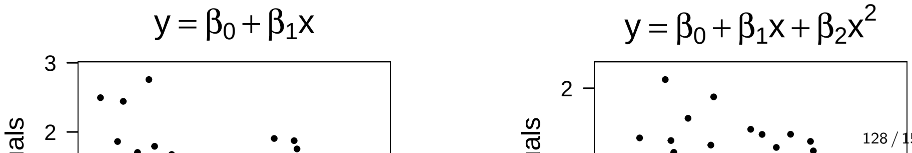




Residuals vs predictors

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

```
res_x <- ggplot(res,
       aes(x = x, y = rd)) +
    geom_point() +
    geom_smooth(se = FALSE) +
    xlab("x") +
    ylab("Std. Deviance Residuals") +
    ggtitle(latex2exp::TeX("$y = \\beta_0 + \\beta_1x$")) +
    mytheme()

res_x2 <- ggplot(res,
                 aes(x = x, y = rd2)) +
    geom_point() +
    geom_smooth(se = FALSE) +
    xlab("x") +
    ylab("Std. Deviance Residuals") +
    ggtitle(latex2exp::TeX("$y = \\beta_0 + \\beta_1x + \\beta_2x^2$")) +
    mytheme()

res_x + res_x2
```



Binomial vs Binary

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:

```r
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 0.15 0.03   0
## 2   2  0.9 0.93   1
## 3   3 0.34 0.13   0
## 4   4 0.67 0.68   0
## 5 ...  ...  ... ...
## 6  27 0.45 0.26   0
## 7  28 0.47  0.3   0
## 8  29 0.11 0.02   0
## 9  30 0.44 0.25   0
```

Or equivalently with a categorical variable:

```r
n <- 15
```

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:

```r
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  10   0  10
## 7  0.9   9   1  10
## 8 0.95  10   0  10
## 9    1  10   0  10
```

With a categorical variable we have essentialy a contingency table:

```r
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)))
```

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:

```r
# 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    0   a
## 5 ... ...
## 6    0   b
## 7    0   b
## 8    0   b
## 9    0   b
```

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

```
##   x nc nf nt
## 1 a  3  7 10
## 2 b  4  6 10
```

Binomial vs Binary: - fitting function in R

```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

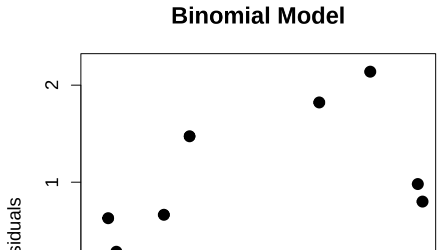
A more relevant difference is about the residual analysis. The binary regression has different residuals compared to the binomial model fitted on the same dataset[6].

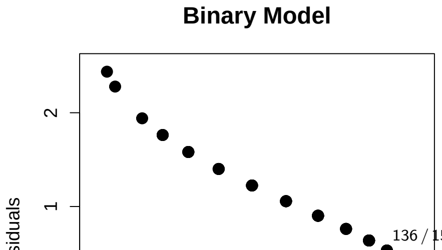
```r
fit_binomial <- glm(nc/nt ~ x, weights = nt, data = dat_binomial, family = binomial(link = "logit"))
summary(fit_binomial)
```

```
##
## Call:
## glm(formula = nc/nt ~ x, family = binomial(link = "logit"), data = dat_binomial,
##     weights = nt)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.34746  -0.57617  -0.09798   0.62605   2.00438
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -4.4896     0.6036  -7.439 1.02e-13 ***
## x             7.9901     1.0260   7.788 6.82e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 157.225  on 20  degrees of freedom
## Residual deviance:  23.543  on 19  degrees of freedom
## AIC: 58.075
##
## Number of Fisher Scoring iterations: 5
```

Binomial vs Binary: residuals and residual deviance

```r
par(mfrow = c(1,2))

plot(fitted(fit_binomial), rstandard(fit_binomial, type = "deviance"),
     xlab = "Fitted", ylab = c("Std. Deviance Residuals"),
     pch = 19,
     lwd = 5,
     cex.lab = 1.2,
     cex.axis = 1.2,
     cex.main = 1.5,
     main = "Binomial Model")

plot(fitted(fit_binary), rstandard(fit_binary, type = "deviance"),
     xlab = "Fitted", ylab = c("Std. Deviance Residuals"),
     pch = 19,
     lwd = 5,
     cex.lab = 1.2,
     cex.axis = 1.2,
     cex.main = 1.5,
     main = "Binary Model")
```





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[7]:

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

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     154.69
## 2       209     288.37 -1  -133.68 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
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     23.543
## 2        20    157.225 -1  -133.68 < 2.2e-16 ***
```

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
- he 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

```r
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          5   1
## 2   1        2          4   0
## 3   1        3          5   1
## 4   1        4          5   0
## 5 ...      ...        ... ...
## 6   1       27          2   1
## 7   1       28          1   1
## 8   1       29          1   1
## 9   1       30          2   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)

```
dat <- data.frame(background = c("math", "chemistry", "art", "sport"))
nt <- c(30, 20, 10, 20)
dat$nc <- rbinom(nrow(dat), nt, 0.7)
dat$nf <- nt - dat$nc
dat$nt <- nt

dat |>
    trim_df()
```

```
##   background nc nf nt
## 1       math 21  9 30
## 2  chemistry 12  8 20
## 3        art  8  2 10
## 4      sport 15  5 20
```

Or the binary version:

```
dat |>
    bin_to_binary(nc, nt) |>
    trim_df()
```

```
##      y background  nc  nf  nt
```

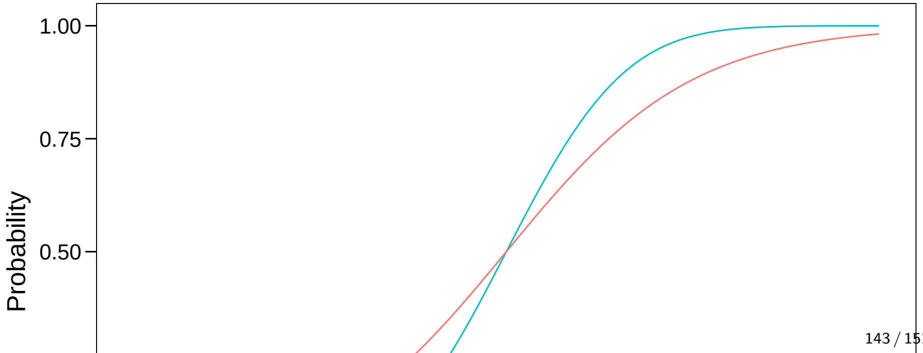
Binomial vs Binary: type of predictors [EXTRA]

- 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 togheter 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 neglegible.



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 comulative normal distribution.

```r
p1 <- 0.7
p2 <- 0.5

qlogis(c(p1, p2)) # log(odds(p1)), logit link
```

```
## [1] 0.8472979 0.0000000
```

```r
qnorm(c(p1, p2)) # probit link
```

```
## [1] 0.5244005 0.0000000
```

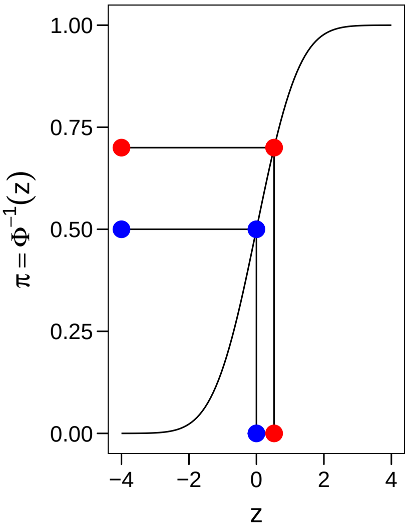
```r
log(odds_ratio(p1, p2)) # ~ beta1, logit link
```

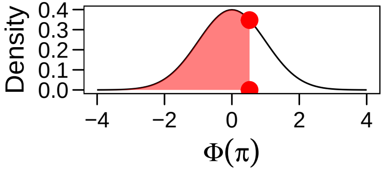
```
## [1] 0.8472979
```

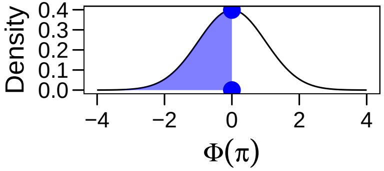
```r
pnorm(p1) - pnorm(p2) # ~beta1, probit link
```

```
## [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 [4]. 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.

```r
d <- 1.5

ggplot() +
    stat_function(fun = dnorm,
                  aes(color = "Noise"),
                  linewidth = 0.8) +
    stat_function(fun = dnorm,
                  aes(color = "Signal"),
                  linewidth = 0.8,
                  args = list(mean = 1.5, sd = 1)) +
    xlim(c(d/2 - 4.5, d/2 + 4.5)) +
    scale_color_manual(values = c("black", "red")) +
    geom_vline(xintercept = d/2, linetype = "dashed") +
    geom_segment(aes(x = 0,
                 xend = d,
                 y = dnorm(0, 0, 1)+0.01,
                 yend = dnorm(d, d, 1)+0.01)) +
    geom_label(aes(x = d/2, y = dnorm(d, d, 1)+0.01),
               label = latex2exp::TeX("d$'$"),
               fill = "lightgreen",
               size = 5) +
```

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 - **specificity** defined for the ROC analysis (*say signal present when the signal is actually absent*).

Binomial GLM - Probit link and Signal Detection Analysis [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 Signal Detection Analysis [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 childern *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 $n = 100$ childern and we want to assess the ability of the clinician detecting the learning disorder:

```
##   say_disorder has_disorder          x
## 1            1            1  1.7854928
## 2            1            1  1.0829877
## 3            1            1  1.4377658
## 4            1            1  0.6543622
## 5            0            1 -0.1596302
## 6            0            1  0.2171808
```

Binomial GLM - Probit link and SDT [EXTRA]

```r
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
##   38   12   36   14
```

```r
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
## 1.289144
```

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

```
##        hit
## 0.06173053
```

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)
##  0.06173053
```

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