Computational Cognitive Science 2019-2020

Tutorial 4: Model Comparison à la XKCD

Our data in this tutorial comes from a visual motion perception task (Karvelis et al., 2018): participants saw either moving dots or no stimuli for 3000 milliseconds, then reported the motion of direction, or an absence of stimuli if no dots were present. The independent variable `RISC` is a scale of schizotypical traits, measured on the healthy participants in the study. The dependent variable `false.detections` is a count of the number of trials in which participants falsely detected (i.e. hallucinated) stimuli — that is, they reported that moving dots were present when no dots were shown on that trial. See the original paper for more details.

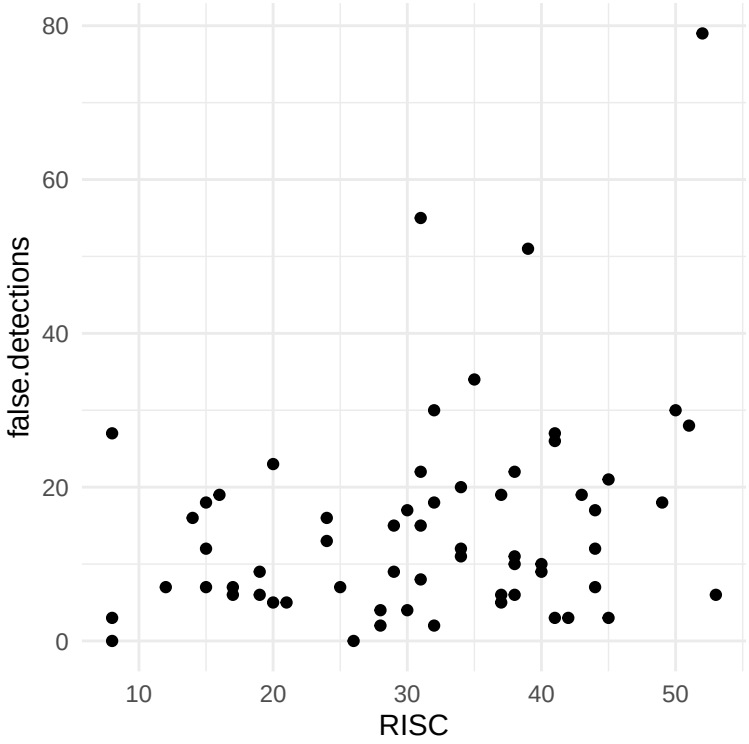
Consider an initial sample of 62 randomly selected participants (perhaps the first batch of subjects tested). We'll load in their data for analysis.

```
data <- read.csv("sample1.csv")
```

Inspired by Randall Munroe's XKCD webcomic (**link**), we will compare the fit of various models to some real-world data. As before, we'll use ggplot to visualize.

```
library(ggplot2)

base_plot <- ggplot(data) + # pass data to ggplot
  aes(RISC, false.detections) + # tell it what to plot on x and y axes respectively
  geom_point() + # add scatterplot layer
  theme_minimal() # nicer-looking style
base_plot
```



If you would like to get the full XKCD feel, you can optionally install the xkcd package for a bit more verisimilitude in plotting! But if you don't want to, no worries, all the later code will run just fine without it.

```
# check for xkcd package, install if missing
xkcd_available <- require("xkcd")
if (!xkcd_available) {
  install.packages("xkcd", repos="http://cran.uk.r-project.org")
  xkcd_available <- require("xkcd")
}

# Note: xkcd_available will be false if the installation didn't work,
# in which case we'll have to fall back to base_plot
if(xkcd_available) {

  # check for xkcd font, install if missing
  if (!('xkcd' %in% fonts())) {
    download.file("http://simonsoftware.se/other/xkcd.ttf",
                  dest="xkcd.ttf", mode="wb")
    system("mkdir  ~/.fonts")
    system("cp xkcd.ttf  ~/.fonts")
    #system("cp xkcd.ttf  /Library/Fonts") # uncomment if you're running on a Mac
    font_import(paths=c("~/.fonts", "/Library/Fonts"), pattern = "[X/x]kcd", prompt=FALSE)
    loadfonts()
  }

  # add xkcd layers to the plot we defined above
  base_plot <- base_plot +
    xkcdaxis(range(data$RISC), range(data$false.detections)) + # make plot axes look xkcd-ish
    theme(text=element_text(family="xkcd", size=14)) # use xkcd font
}

# Here it is again, in XKCD style
base_plot
```

Nested models

First, we'll fit the classic linear model: $y = \beta_1 x + \beta_0$.

```
# for more details on R's linear model function:
# ?lm
lm_fit <- lm(false.detections ~ RISC, # formula: y ~ x
             data=data)
# intercept term is assumed, doesn't need to be specified
# so actual formula is y ~ x + 1

summary(lm_fit)
```

```
##
## Call:
## lm(formula = false.detections ~ RISC, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -16.838  -7.461  -2.749   4.469  56.705
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
```

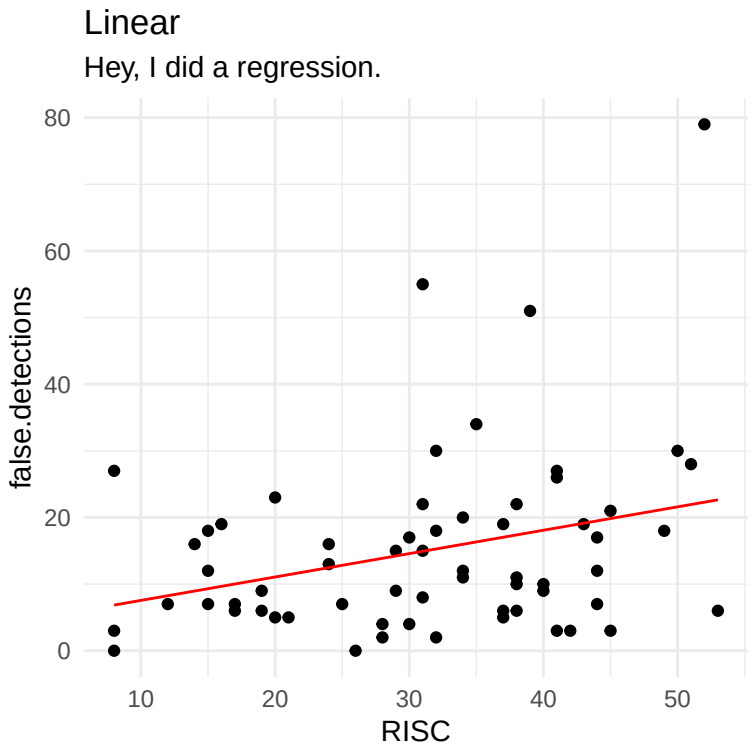
```
## (Intercept)   4.0445     4.8104   0.841   0.4038
## RISC          0.3510     0.1439   2.438   0.0177 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 13.26 on 60 degrees of freedom
## Multiple R-squared:  0.09016,    Adjusted R-squared:  0.075
## F-statistic: 5.946 on 1 and 60 DF,  p-value: 0.01773
```

Look over the summary output. What are the estimated values of the coefficients β_0 and β_1? What are the standard errors? Does this look like a reasonable fit to you?

We can define a helper function to plot the model's predictions.

```
plot_preds <- function(preds, title="", subtitle="") {
  (base_plot + geom_line(aes(y=preds), col="red") + ggtitle(title, subtitle))
}

plot_preds(fitted(lm_fit), # 'fitted' extracts model's predictions for training data
           "Linear", "Hey, I did a regression.")
```



As noted in the original comic, we can also just fit the intercept term, β_0, to get a linear model with no slope.

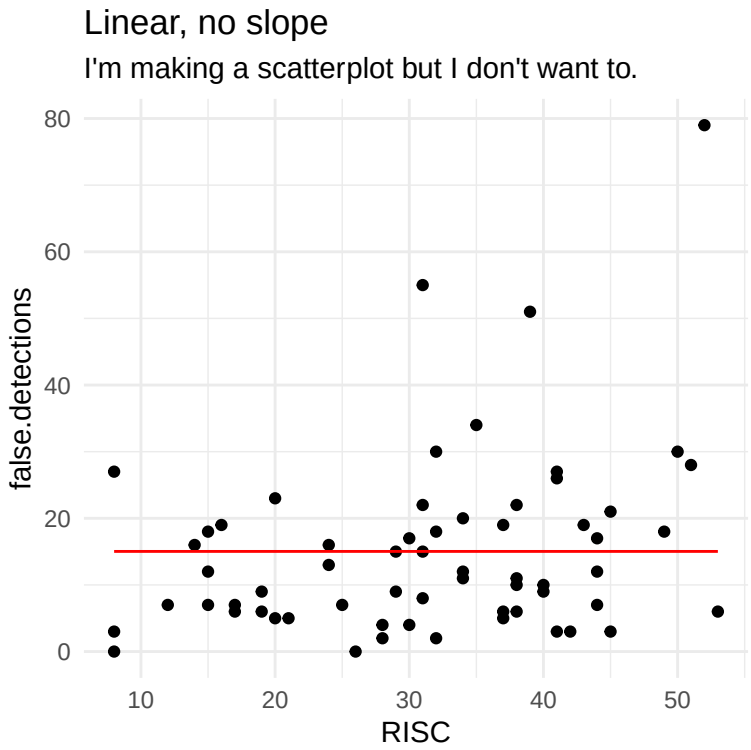
```
intercept_fit <- lm(false.detections ~ 1, data=data)

summary(intercept_fit)
```

```
##
## Call:
## lm(formula = false.detections ~ 1, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
```

```
## -15.032  -9.032  -3.532   3.968  63.968
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   15.032      1.751   8.587 4.33e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 13.78 on 61 degrees of freedom
```

```
plot_preds(fitted(intercept_fit),
           "Linear, no slope", "I'm making a scatterplot but I don't want to.")
```



Exercise: Modify the model-fitting code above to fit a quadratic function, i.e. an order-2 polynomial: $y = \beta_2 x^2 + \beta_1 x + \beta_0$.

```
quad_fit <-  # TODO *YOUR CODE HERE*

summary(quad_fit)

plot_preds(fitted(quad_fit), "Quadratic", "I wanted a curved line,\nso I made one with math.")
```

As these are all nested models, we can compare them using the likelihood ratio test in R's `anova` function.

```
anova(intercept_fit, lm_fit,
      #quad_fit, # TODO uncomment once you've defined quad_fit
      test="LRT") # specify likelihood ratio test
```

```
## Analysis of Variance Table
##
## Model 1: false.detections ~ 1
## Model 2: false.detections ~ RISC
```

```
##   Res.Df   RSS Df Sum of Sq Pr(>Chi)
## 1     61 11590
## 2     60 10545  1      1045  0.01475 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Exercise: Can you think of any reasons you might not want to use the likelihood ratio test? Are there other approaches you could use to compare nested models?

AIC & BIC

A more general approach which does not require models to be nested is to use AIC or BIC.

```
models = list(intercept_fit, lm_fit
              #, quad_fit # TODO uncomment once you've defined quad_fit
              )
model_AIC = AIC(intercept_fit, lm_fit
                #, quad_fit # TODO uncomment once you've defined quad_fit
                )

cbind(model_AIC,
      BIC = sapply(models, BIC),
      log_lik = sapply(models, logLik))
```

```
##               df      AIC      BIC   log_lik
## intercept_fit  2 504.2554 508.5096 -250.1277
## lm_fit         3 500.3970 506.7784 -247.1985
```

Exercise: How do AIC and BIC estimate model complexity? What's the difference between the two? Are there cases when you can't use these metrics?

Held out set

An *even better* approach to model comparison is to directly predict — as we're mainly interested in a model's ability to generalize to unseen data, evaluating on a sequestered set lets us look at prediction directly, without even considering model complexity.

Let's load in the data from the remaining 20 participants (perhaps the second batch tested), and see how well our models predict their behavior. For simplicity, we'll use mean squared error to evaluate performance here, but in most settings log predictive density would be preferred (Gelman et al., 2014).

```
sample2 <- read.csv("sample2.csv")
```

Exercise: Calculate the mean squared error (MSE) for each model's predictions on the new data. (Hint: you can use the function `predict`; for example, `preds <- predict(lm_fit, sample2)` gets the linear model's predictions for `sample2`, which you can then use to calculate MSE.) Which one has the lowest mean squared error on the new data?

Cross-validation

We can use *cross-validation* to get a more robust comparison of different models' ability to generalize to unseen data, by taking all the data available and systematically using different subsets to estimate test error.

```
data <- rbind(data, sample2) # combine all our data
```

```
n_splits = 6 # specify number of splits to partition
```

```
# assign each row a split
# this assumes the rows are randomly ordered
splits = rep_len(1:n_splits, length.out=nrow(data))

# you can use `splits` to index your data frame
# for example, to test on split 1:
# training_data <- data[splits != 1,]
# test_data <- data[splits == 1,]
```

Exercise: Cross-validate the intercept, linear, and quadratic models using the splits defined above. Which model has the lowest average test error across splits? Do you get different results if you try different numbers of splits (e.g. by setting `n_splits` to another value)?

References:

Karvelis, Povilas, Aaron R Seitz, Stephen M Lawrie, and Peggy Seriès (2018). "Autistic Traits, but Not Schizotypy, Predict Increased Weighting of Sensory Information in Bayesian Visual Integration." ELife 7. https://doi.org/10.7554/eLife.34115.

Gelman, Andrew, Jessica Hwang, and Aki Vehtari (2014). "Understanding Predictive Information Criteria for Bayesian Models." Statistics and Computing 24:6: 997–1016. https://doi.org/10.1007/s11222-013-9416-2.