Performing Fractional Polynomials in R

Lena Kristin Bache-Mathiesen

Contents

Introduction

This document is intended as a guide for modeling the relationship between a measure of training load and injury in sports research, using fractional polynomials (FP). The examples will go through standard logistic regression, and later a mixed effects logistic regression model.However, the steps to model training load using FP in a Poisson model and other regression models are the same.

Preparation

First, load required packages.

```
library(tidyverse) # for creating figures, and for a self-made function we'll need later on
library(rlang) # for creating functions with tidyverse syntax
library(mfp) # the mfp package has functions for automatic determination of FP
library(lme4) # package for mixed model functions
library(sjPlot) # for plotting predicted values with splines
library(ggeffects) # for model predictions with splines
library(clubSandwich) # for cluster-robust confidence intervals
```

Next step is to load your data. Here we used the d_example_guide.rds data available from the GitHub repository. It is simulated from football data for the example to be reproducible. The object "d" in the r-code below can be replaced with your own data. At the minimum, your data should have:

- one column for load, measured in any metric (such as sRPE, GPS measures or ACWR)
- one column for injury, must either be a logical variable (TRUE/FALSE) or coded (0 for no injury, 1 for injury) to work in our logistic regression model.
- one column for athlete ID, coupling the load values and injuries to the right person

```
d = readRDS("d_example_guide.rds")
```

```
# For model predictions to be calculated by
# the ggeffects package, categorical variables
# must be factors, not the character class
d = d %>% mutate(p_id = as.factor(p_id))
```

Standard Logistic Regression

Fitting the model

For a regular logistic regression model, we can write:

```
fit_logistic = glm(injury ~ load, data = d)
```

Running `fit_logistic` or `summary(fit_logistic)` will provide us the results and information from the fit.

Fitting load with fractional polynomial terms, we can run the code below. Note that it may take a little time.

```
fit_fp = mfp(injury ~ fp(load), data = d, family = "binomial")
```

The function `fp()` is from the `mfp`package. The model searches for the best fit of a number of possible polynomial transformations. It uses a backwards selection process.

Running `summary()` of the fit will provide information of how many polynomials terms were added (of 1 or 2), and to what power they were chosen. Estimate is the beta-value and represents the logodds. The p-value indicates significance of each polynomial term.

By saving the object like this:

```
fit_summary = summary(fit_fp)
```

It's possible to access different data from the fit For example, the aic, or the coefficients, to use them later:

```
fit_summary$aic
```

```
## [1] 14777.15
```

```
fit_summary$coefficients
```

```
##                             Estimate   Std. Error   z value     Pr(>|z|)
## (Intercept)            3.3551343943 0.0437397223  76.70681 0.000000e+00
## I(((load + 1)/100)^1) -0.4935883599 0.0110508208 -44.66531 0.000000e+00
## I(((load + 1)/100)^3)  0.0009313095 0.0000840651  11.07843 1.596439e-28
```

To obtain Odds Ratios, we can run:

```
exp(coef(fit_fp))
```

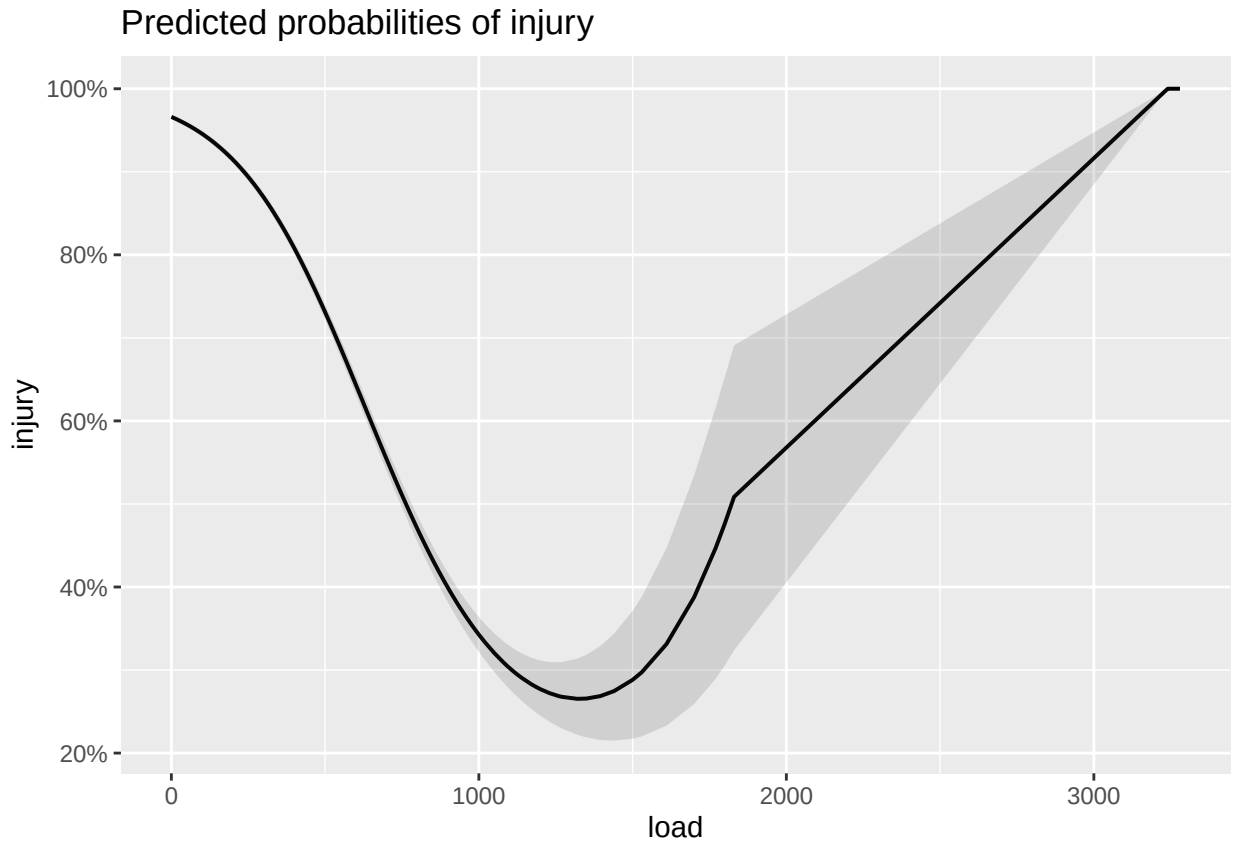
```
## Intercept    load.1    load.2
## 28.649454  0.610432  1.000932
```

Visualization

With fractional polynomials, we can interpret our results numerically, but it may still be useful to create a figure of the predictions to see the relationship shape.

The `sjPlot` package has a handy function to create a simple form of `ggplot2`-plot. Unfortunately, the syntax is not compatible with that in the `mfp`-package we therefore have to write out our model manually by checking the formula in the object `fit_fp` we created previously.

```
fit_fp_manual = glm(injury ~ load + I(((load + 1)/100)^3), data = d, family = "binomial")
plot_model(fit_fp_manual, type = "pred", terms = "load [all]")
```



Mapping the load values to the predicted probabilities (figure above) showed that the polynomials modelled a U-shaped relationship for the example data.

Mixed Effects Regression Model

Fitting the model

A mixed effects logistic regression model is a bit more complicated most notably, because running a general linear mixed model (GLMM) isn't part of base R. Here we use the `lme4` package for mixed models. Not everything in base R or other packages is compatible with the `lme4` package. Most critically, the `mfp()` package for running fractional polynomials are not compatible with the `lme4` package.

For a mixed model with binomial distribution and random intercept per athlete one can run:

```
fit_mixed = glmer(injury ~ load + (1 | p_id), family = "binomial", data = d)
```

For a mixed model with binomial distribution, random intercept and random slope per athlete

```
fit_mixed_slope = glmer(injury ~ load + (load | p_id), family = "binomial", data = d)
```

Since running the `mfp()`-function won't work inside `glmer()`, we need another solution. One solution would be manually model every possible incarnation of an FP2 model and manually determine the best fit.

Here we've made an automatized solution. Below is a function, `glmer_fp()` that searches for the best FP fit in a standard regression model with the `mfp`-function we used earlier. The FP-formula that was found to be

the best one is then extracted from the standard model and run in a random effects model using `glmer()`. the default is a logistic regression (binomial distribution of outcome measure).

The function was created using `tidyverse`, `rlang` (which is why we loaded these at the top), the `lme4` and `mfp` package.

```
# the function has the following arguments:
# d                 The dataset with the load and injury variable
# injury            The variable used to denote injury.
#                   Can be heealth problems or any other definition of injury.
# load              The load variable in the dataset.
# rdm_effect        The random effect term.
#                   Must be surrounded by quotes "". Examples are "(1|your_id_variable)"
#                   for a random intercept and "(load|your_id_variable)"
#                   for a random slope + random intercept.
# family            Determine the model family. The defualt is binomial,
#                   meaning the models will run logistic regression.
#                   can be sett to "poisson" or other alternatives, see ?glmer()
# for a multivariable model, extra covariates will have to be added
# using + after fp(!!load), i.e. !!injury ~ fp(!!load) + sex + age,
# but the names must match those in the data
# and the function will no longer be general
glmer_fp = function(d, injury, load, rdm_effect, family = "binomial"){
  injury = enexpr(injury)
  load = enexpr(load)

  # run mfp to find best FP terms
  prox_fit = eval_bare(expr(mfp(!!injury ~ fp(!!load), data = d, family = family)))
  fp_form = prox_fit$formula

  # automatically use that formula in a random effects model
  formula_start = paste0("!!",fp_form[2], " ", fp_form[1])
  formula = paste0(formula_start, " ", fp_form[3], " + ",rdm_effect,"")
  glmm_fit = eval_bare(expr(glmer(as.formula(formula), family = family, data = d)))
  glmm_fit
}
```

An example of using the function with a random intercept and random slope:

```
fit_mixed = glmer_fp(d, injury, load, "(load|p_id)")
```

An example of using the function with a random intercept only:

```
fit_mixed_intercept = glmer_fp(d, injury, load, "(1|p_id)")
```

If both were able to converge, we can determine best fit with AIC:

```
AIC(fit_mixed)
```

```
## [1] 14744.94
```

```
AIC(fit_mixed_intercept)
```

```
## [1] 14742.65
```

As seen above, the random intercept model had the lowest AIC and therefore the better fit.

Visualization

We can visualize in the same manner as the model without random effects Again, we have to write out our fit manually to do so.

```
fit_mixed_manual = glmer(injury ~ load + I(((load + 1)/100)^3) + (1|p_id),
                         data = d,  family = "binomial")
```

Since this is a mixed model, we can calculate cluster-robust confidence intervals which take into account the uncertainty stemming from random effects variance. We can do this by calculating the predictions with `ggpredict()` and specifying the function for obtaining the variance-covariance matrix from the model. We use `vcovCR()` from the `clubSandwich` package and specify the clusters in a list in the `vcov.args` argument.

```
preds = ggpredict(
  fit_mixed_manual,
  "load [all]",
  vcov.fun = "vcovCR",
  vcov.type = "CR0",
  vcov.args = list(p_id = d$p_id),
  type = "re.zi"
)
plot(preds)
```