Multivariable Fractional Polynomials with Extensions

Edwin Kipruto Michael Kammer Patrick Royston Willi Sauerbrei

June 21, 2023

Contents

1 Introduction to mfp2 package

`mfp2` is a package that selects the MFP model. In addition, it has the ability to model a sigmoid relationship between `x` and an outcome variable `y` using the ACD transformation proposed by Royston (2016). The package offers three options for variable and function selection: p-value, Akaike information criterion (AIC), and Bayesian information criterion (BIC). Furthermore, it provides functions for prediction and plotting. Currently, the package implements linear, logistic, Poisson, and Cox regression models. However, the package is designed in such a way that it can easily incorporate other generalized linear models or parametric survival models.

The main function, `mfp2()`, implements both MFP and MFP with ACD transformation. It offers two interfaces for input data. The first interface allows direct input of the predictor matrix `x` and the outcome vector `y`. The second interface uses a formula object in conjunction with a data.frame, similar to the `glm()` function with slight modifications. Both interfaces are equivalent in terms of functionality.

The authors of `mfp2` are Edwin Kipruto, Michael Kammer, Patrick Royston, and Willi Sauerbrei, with contribution from Gregory Steiner and Georg Heinze. The R package is maintained by Edwin Kipruto, while the STATA version of `mfp` is maintained by Patrick Royston.

This vignette describes basic usage of `mfp2` in R. There are additional vignettes available that will further enhance your understanding on the mfp2 package.

- "Introduction to Multivariable Fractional Polynomial" provides an overview of multivariable fractional polynomials.

1.1 Estimation algorithm

The estimation algorithm employed in `mfp2` sequentially processes the predictors using a back-fitting approach. It calculates the p-values of each predictor using the likelihood ratio test, assuming linearity. Subsequently, the predictors are arranged based on these p-values. By default, the predictors are arranged in order of decreasing statistical significance. This ordering aims to prioritize modeling relatively important variables before less important ones. This approach may help mitigate potential challenges in model fitting arising from collinearity or more generally, the presence of "concurvity" among the predictors (stata??). Although alternative options for predictor ordering are available, we prefer the default option.

If a predictor contains nonpositive values, the program by default shifts the location of the predictor x to ensure positivity. In addition, it scales the shifted predictor before the first cycle of the algorithm. For more information on shifting and scaling, please refer to section xx.

At the initial cycle, the best-fitting FP function for the first variable (after ordering) is determined, with all the other variables assumed to be linear. The functional form (but not the estimated regression coefficients) is kept, and the process is repeated for the other variables. The first iteration concludes when all the variables have been processed in this way. The next cycle is similar, except that the functional forms from the initial cycle are retained for all variables except the one currently being processed

A variable whose functional form is prespecified to be linear is tested for exclusion within the above procedure when its nominal p-value is less than 1 or argument `keep = FALSE`; otherwise, it is included. Updating of FP functions and candidate variables continues until the functions and variables included in the overall model do not change (convergence). Convergence is usually achieved within 1–5 cycles.

1.2 Installation

To install the `mfp2` package, enter the following command in the R console:

```
install.packages("mfp2")
```

1.3 Quick Start

The purpose of this section is to provide users with a comprehensive understanding of the `mfp2` package. We will provide a concise overview of its key functions and resulting outputs. We will delve into each function in detail, highlighting their specific use. This will provide users with a deeper understanding of the package's functionality. The package includes a built-in dataset that is specifically designed for analysis within the `mfp2` framework.

To begin, let's load the `mfp2` package:

```
library(mfp2)
```

1.4 Linear Regression

The default `family` in `mfp2` package is `Gaussian`, which fits a Gaussian linear model. In this section, we will demonstrate how to fit this model. We will use the prostate cancer data (Stamey et al., 1989) included in our package. The dataset contains seven predictors (six continuous variables and one binary variable) and a continuous outcome variable (log prostate-specific antigen (lpsa)) of 97 patients with prostate cancer. Our aim is to determine whether non-linear functional relationships exist between the predictors and the outcome variable.

Load the `prostate` dataset from the `mfp2` package and display the first few rows of the dataset

```
# Load prostate data
data("prostate")
head(prostate)
#> # A tibble: 6 x 9
#>   obsno   age   svi pgg45 cavol weight   bph    cp   lpsa
#>   <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl> <dbl> <dbl>  <dbl>
#> 1     1    50     0     0 0.560   16.0  0.25  0.25 -0.431
#> 2     2    58     0     0 0.370   27.7  0.25  0.25 -0.163
#> 3     3    74     0    20 0.600   14.7  0.25  0.25 -0.163
#> 4     4    58     0     0 0.300   26.6  0.25  0.25 -0.163
#> 5     5    62     0     0 2.12    31.0  0.25  0.25  0.372
#> 6     6    50     0     0 0.350   25.2  0.25  0.25  0.765

# create predictor matrix x and numeric vector y
x <- as.matrix(prostate[,2:8])
y <- as.numeric(prostate$lpsa)
```

The command loads a dataframe from the **R** data archive since the `mfp2` package is already loaded. We create a matrix `x` and a numeric vector `y` from the dataframe.

1.4.1 Fitting MFP Models Using Default and Formula Interface

The default interface of `mfp2()` requires a matrix of predictors `x` and a numeric vector of response `y` for continuous outcomes. If you have one predictor, make sure you convert it into a matrix with a single column.

We demonstrate how to fit the Gaussian linear model using the default interface of `mfp2()` with default parameters.

```
fit <- mfp2(x, y)
```

The fit is an object of class `mfp2` that inherits from `glm` and `lm`. This means that `mfp2` inherits properties and methods from `glm` and `lm`, which allows the `mfp2` to utilize methods and functions specific to `glm` and `lm`. Various methods are defined for extracting components from the `mfp2` object, such as coef, print, summary, fracplot, and predict. These methods allow users to access different components of the mfp2 object, including coefficient estimates, plotting functionalities, and predictions.

To fit the same model using the formula interface, we use the `fp()` function.

```
fit <- mfp2(lpsa ~ fp(age) + svi + fp(pgg45) + fp(cavol) + fp(weight) + fp(bph) + fp(cp), data = prosta
```

The main distinction between the `mfp2()` and `glm()` functions in **R** is the inclusion of the `fp()` function within the formula. The presence of the `fp()` function in the formula indicates that the variables included

within it should undergo fractional polynomial (FP) transformation, provided that the degree of freedom (df) is not equal to 1. A df of 1 indicates a linear relationship, which does not require transformation. Note that df is an argument in the `fp()` function. For more details on the fp() function, please refer to section xx

The variable `svi` is a binary variable and is therefore not passed to the `fp()` function. This is because binary or factor variables do not undergo FP transformation. If a binary variable is passed to the `fp()` function, the program will automatically set the df to 1, treating the variable as linear. However, passing a factor variable to the `fp()` function will result in an error. For more details on how `mfp2` handles factor variables, refer to section XX.

1.4.2 Shifting and Scaling of Predictors

Fractional polynomials are defined only for positive variables due to the use of logarithms and other powers such as square root. Thus, `mfp2()`function estimates shifting factors for each variables to ensure positivity. The function `find_shift_factor()`, used internally by `mfp2`, automatically estimates shifting factors for each continuous variables. The formula used to estimate the shifting factor for a variable, say `x1`, is given by:

$$shift_{x1} = \gamma - min(x1)$$

where $min(x1)$ is the smallest observed value of `x1`, while γ is the minimum increment between successive ordered sample values of `x1`, excluding 0 (Royston and Sauerbrei,2008). The new variable $x1' = x1 + shift_{x1}$ will then be used by `mfp2()` in estimating the FP powers.

For example, to estimate shifting factors for predictor matrix `x` from prostate data in R, you can run the following code:

```
# minimum values for each predictor
apply(x, 2, min)
#>    age    svi  pgg45  cavol weight    bph     cp
#>  41.00   0.00   0.00   0.26  10.75   0.25   0.25

# shifting values for each predictor
apply(x, 2, find_shift_factor)
#>    age    svi  pgg45  cavol weight    bph     cp
#>      0      0      1      0      0      0      0
```

We see that among the continuous variables, only the variable `pgg45` is shifted by a factor of 1, which is attributed to its minimum value being 0. Even though the variable `svi` also has a minimum value of 0, it is not shifted because its a binary variable. The user can manually set the shifting factors for each variable in `mfp2()` function.

If the values of the variables are too large or too small, it is important to scale the variables to reduce the chances of numerical underflow or overflow which can lead to inaccuracies and difficulties in estimating the model. Scaling can be done automatically or by directly specifying the scaling values for each variables so that the magnitude of the `x` values are not too extreme. By default scaling factors are estimated by the program as follows.

After adjusting the location of `x` (if necessary) so that its minimum value is positive, creating `x'` automatic scaling will divide each value of `x'` by 10^p where the exponent `p` is given by

$$p = sign(k) \times floor(|k|) \quad \text{where} \quad k = log_{10}(max(x') - min(x'))$$

The `mfp2()` function uses this formula to scale `x` matrix, and the scaling process is implemented through the `find_scale_factor()` function. The following R code demonstrates the estimation of scaling factors for `x`. From the output below, we see that the variables `age`, `cavol`, and `cp` have scaling factors of 10 each, while the variables `pgg45` and `weight` have scaling factors of 100 each. Each variable will be divided by its corresponding scaling factor. A scaling factor of 1 implies no scaling.

```r
# shift x if nonpositive values exist
shift <- apply(x, 2, find_shift_factor)
xnew <- sweep(x, 2, shift, "+")

# scaling factors
apply(xnew, 2, find_scale_factor)
#>    age    svi  pgg45  cavol weight    bph     cp
#>     10      1    100     10    100      1     10
```

To manually enter shifting and scaling factors, the `mfp2()` function provides the `shift` and `scale` arguments. In the default usage of `mfp2()`, a vector of shifting or scaling factors, with a length equal to the number of predictors, can be provided. In the formula interface, shifting or scaling factors can be directly specified within the`fp()` function. Below is an example to illustrate this:

```r
# Default interface
mfp2(x,y, shift = c(0, 0, 1, 0, 0, 0, 0), scale = c(10, 1, 100, 10, 100, 1, 10))

# Formula interface
mfp2(lpsa ~ fp(age, shift = 0, scale = 10) + svi + fp(pgg45, shift = 1, scale = 100) + fp(cavol, shift =
            data = prostate)
```

In the default interface, each variable in the `x` matrix is assigned a shifting and scaling factor based on their respective positions. For instance, the first variable in the column of `x`, which is `age`, is assigned a shifting factor of 0 and a scaling factor of 10. The second variable, `svi`, is assigned a shifting factor of 0 and a scaling factor of 1, and so on.

1.4.3 Setting degrees of freedom for each variable

The degrees of freedom (df) for each predictor (excluding the intercept) are twice the degrees of freedom of the FP. For instance, if the maximum allowed complexity of variable x_1 is a second-degree FP (FP2), then the degrees of freedom assigned to this variable should be 4. The default `df` is 4 for each predictor.

After assigning the default degrees of freedom to each variable, the program proceeds and overrides the default value based on the number of unique values for a given variable. The rules for overriding the default degrees of freedom are as follows:

- If a variable has 2-3 distinct values, it is assigned `df` = 1 (linear).
- If a variable has 4-5 distinct values, it is assigned `df` = min(2, default).
- If a variable has 6 or more distinct values, it is assigned `df` = default.

These rules ensure that the appropriate `df` are assigned to variables. For instance, it is not sensible to fit an FP2 function to a variable with only 3 distinct values.

The following code illustrates how to set different `df` for each variable. In the default interface, the `df` of the binary variable `svi` is explicitly set to 1, while In the formula interface, there is no need to specify the `df`, as the program automatically assigns `df = 1` for binary variables.

If the user attempts to enter `df = 4` for `svi` in the default interface, the program will reset the `df` to 1 and issue a warning.

```r
# Default Interface
mfp2(x,y, df = c(4, 1, 4, 4, 4, 4, 4))

# Formula Interface
mfp2(lpsa ~ fp(age, df = 4) + svi + fp(pgg45, df = 4) + fp(cavol, df = 4) + fp(weight, df = 4) +
       fp(bph, df = 4) + fp(cp, df = 4),data = prostate)
```

If the user does not explicitly assign `df` to variables, the program will automatically assign a `df = 4` to each variable. It is important to note that even if a continuous variable is not passed to the `fp()function in the formula interface, the`df‘ of that variable will still be set to the default value and will later be adjusted based on the unique values. The following three examples are equivalent:

```r
# Default Interface
mfp2(x,y)

# Formula Interface
mfp2(lpsa ~ fp(age) + svi + fp(pgg45) + fp(cavol) + fp(weight) +
       fp(bph) + fp(cp), data = prostate)

# Formula Interface but `cp` not passed to the fp() function
mfp2(lpsa ~ fp(age) + svi + fp(pgg45) + fp(cavol) + fp(weight) +
       fp(bph) + cp, data = prostate)
```

1.4.4 Tuning parameters for MFP

The two key components of MFP are variable selection with backward elimination (BE) and function selection for continuous variables through the function selection procedure (FSP). These components require two nominal significance levels: α_1 for variable selection with BE and α_2 for comparing the fit of the functions within the FSP. The choice of these significance levels strongly influences the complexity and stability of the final model. While it is possible to use the same nominal significance level for both components, they can also differ based on the aims of the analysis.

The `mfp2()` function has an argument called `criterion`, which allows the user to specify the criteria for variable and function selection. The default criterion is `pvalue`, which enables the user to set the two nominal significance levels. Please refer to section section 1.4.4.1 for instructions on setting the nominal significance levels.

Information criteria, such as the Akaike information criterion (AIC) and the Bayesian information criterion (BIC), have been proposed for selecting models fitted on the same data. The `mfp2` package offers an alternative approach to variable and function selection by utilizing information criteria. In the MFP framework, each predictor is evaluated univariately while accounting for the other predictors within an overarching backfitting algorithm. This algorithm iteratively assesses each predictor and selects the model (null, linear, FP1, FP2, etc.) with the minimum AIC or BIC for each variable.

The “null” model refers to a model without the predictor of interest. A “linear” model assumes a linear relationship with the outcome variable, while an FP1 model assumes a non-linear relationship using the FP1 function. The `criterion` argument allows users to specify whether they want to use AIC or BIC criteria for both variable and function selection. If AIC or BIC is selected as the criterion, the nominal significance levels set through the `select` and `alpha` arguments will be ignored. For details on using AIC and BIC in variable and function selection, please refer to section 1.4.4.2.

1.4.4.1 Nominal significance levels The `mfp2()` function has the arguments `select` and `alpha` for setting the nominal significance level for variable selection by BE and for testing between FP models of

different degrees, respectively. It is important to note that when using these arguments, you should ensure that the `criterion` is set to `"pvalue"` to correctly use the specified significance levels.

For variable selection, a significance level can be set using the `select` argument. A value of 1 (`select = 1`) for all variables forces them all into the model. Setting the nominal significance level to be 1 for a given variable forces it into the model, leaving others to be selected or not. A variable is dropped if its removal leads to a nonsignificant increase in deviance

On the other hand, the `alpha` argument is used to determine the complexity of the selected FP function. A value of 1 (`alpha = 1`) will choose the most complex FP function permitted for a given variable. For example, if FP2 is the most complex function allowed for variable x1, setting alpha = 1 will select the FP2 function.

The rules for setting `select` and `alpha` are the same as those for setting `df` (see section 1.4.3). The following R codes shows how to set equal nominal significance levels for variable and function selection ($\alpha_1 = alpha_2 = 0.05$) for each variable and produces identical results. Setting different nominal significance levels is straightforward. Simply replace the value 0.05 with the desired significance level of your choice.

```
# Default Interface
mfp2(x,y, select = rep(0.05, ncol(x)), alpha = rep(0.05, ncol(x)))

# Formula Interface
mfp2(lpsa ~ fp(age, select = 0.05) + svi + fp(pgg45, select = 0.05) + fp(cavol, select = 0.05) +
       fp(weight, select = 0.05) + fp(bph, select = 0.05) + fp(cp, select = 0.05),
     select = 0.05, alpha = 0.05, data = prostate)
```

In the formula interface, binary variables such as `svi` that are not passed in the `fp()` function utilize the global `select` argument. In our example, the global parameter is set to `select = 0.05`. If several binary variables exist in the model, the global parameters will be used for all of them. However, if specific parameters need to be set for individual binary variables, the user can use the `fp()`function. In summary, if a variable is not passed through the `fp` function, it will utilize the global parameters.

Suppose we want to force the variables “age” and “svi” in the model. To achieve this, we have two options:

- Set `select = 1` specifically for these variables: By setting `select = 1` for `age` and `svi` in the `mfp2()` function, we force these variables to be included in the model.

- Use the `keep` argument: Alternatively, we can use the `keep` argument, which resets the nominal significance levels for `age` and `svi` to 1 if they are different from 1. This ensures that these variables are retained in the model.

```
#------Default Interface
# Set select to 1 for age and svi
mfp2(x,y, select = c(1, 1, 0.05, 0.05, 0.05, 0.05, 0.05), alpha = rep(0.05, ncol(x)))

# use keep argument
mfp2(x,y, select = c(0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05), alpha = rep(0.05, ncol(x)),
     keep = c("age", "svi"))

#-------Formula Interface
# use fp() function and set select to 1 for age and svi
mfp2(lpsa ~ fp(age, select = 1) + fp(svi, df = 1, select = 1) + fp(pgg45, select = 0.05) +
       fp(cavol, select = 0.05) + fp(weight, select = 0.05) + fp(bph, select = 0.05) + fp(cp, select = 0
     select = 0.05, alpha = 0.05, data = prostate)
```

```
# use keep argument
mfp2(lpsa ~ fp(age, select = 0.05) + svi + fp(pgg45, select = 0.05) + fp(cavol, select = 0.05) +
       fp(weight, select = 0.05) + fp(bph, select = 0.05) + fp(cp, select = 0.05),
     select = 0.05, alpha = 0.05, keep = c("age", "svi"),  data = prostate)
```

1.4.4.2 Information criterion Instead of using nominal significance levels (α_1, α_2) for variable and function selection, an alternative approach is to directly utilize the AIC or BIC. This can be achieved by setting the `criterion` argument to either "aic" or "bic". Additionally, if there is a need to force certain variables, such as "age" and "svi", into the model, the `keep` argument can be used.

The following R code demonstrates how to implement this approach using both the default and formula interfaces, as well as how to force specific variables into the model:

```
# Default Interface
mfp2(x,y,criterion = "aic", keep = c("age", "svi"))

# Formula Interface
mfp2(lpsa ~ fp(age) + fp(svi, df = 1) + fp(pgg45) + fp(cavol) + fp(weight) + fp(bph) +  fp(cp),
     criterion = "aic", keep = c("age", "svi"), data = prostate)
```

1.4.5 Model comparison tests

The FSP in `mfp2()` function compares various models for the variable of interest. For instance, if the most complex allowed FP function is FP2, the FSP will compare the best FP2 model with the NULL model, the best FP2 model with the Linear model, and the best FP2 model with the best FP1 model, when the `criterion = "pvalue"`. The deviance for each model (NULL, Linear, FP1, and FP2) and their corresponding differences and p-values will be calculated.

When comparing Gaussian models, the mfp2() function provides two options for calculating p-values: the F-test and the Chi-square test. The Chi-square test is the default option. For other model families like Cox or logistic regression models, the Chi-square test is used. For more detailed information, please refer to page 23 of the paper available at this link: https://www.stata.com/manuals/rfp.pdf

To use the F-test in **R**, we can set the `ftest` argument to TRUE (`ftest = TRUE`), as demonstrated in the example below. Conversely, to use the Chi-square test, set `ftest = FALSE`.

```
# Default Interface
mfp2(x,y,criterion = "pvalue", ftest = TRUE)

# Formula Interface
mfp2(lpsa ~ fp(age) + svi + fp(pgg45) + fp(cavol) + fp(weight) + fp(bph) +  fp(cp),
     criterion = "pvalue", ftest = TRUE, data = prostate)
```

Please note that the p-values reported by the `mfp` program in `Stata` for Gaussian models are solely based on the F-test. If you intend to compare the results between the two software packages, it is crucial to ensure that `ftest = TRUE` is set in R.

However, it's important to be aware that the older version of the `mfp` package in R utilizes the chi-square test for all model types. Therefore, when comparing the results between the two R packages, the user must set`ftest = FALSE`.

1.4.6 Fraction Polynomial Powers

1.4.7 Explanation of output from model-selection algorithm

1.5 Logistic Regression

1.6 Poisson regression

1.7 Survival data

We illustrate two of the analyses performed by Sauerbrei and Royston (1999). We use brcancer.dta, which contains prognostic factors data from the German Breast Cancer Study Group of patients with node-positive breast cancer. The response variable is recurrence-free survival time (rectime), and the censoring variable is censrec. There are 686 patients with 299 events. We use Cox regression to predict the log hazard of recurrence from prognostic factors of which five are continuous (x1, x3, x5, x6, x7) and three are binary (x2, x4a, x4b). Hormonal therapy (hormon) is known to reduce recurrence rates and is forced into the model. We use mfp to build a model from the initial set of eight predictors by using the backfitting model-selection algorithm. We set the nominal p-value for variable and FP selection to 0.05 for all variables except hormon, which it is set to 1: