pim: An R package for fitting probabilistic index models

Jan De Neve

December 3, 2012

Contents

1 Introduction

This document explains and illustrates how the `pim`-package can be employed to fit a Probabilistic Index Model (PIM). We refer to Thas et al. (2012) for a detailed overview

on PIMs. If (Y, X) and (Y', X') are i.i.d. then a PIM is defined as

$$\mathrm{P}\{Y \preccurlyeq Y' | X, X'\} = m(X, X'; \beta) = g^{-1}(Z^T\beta) \quad \text{for} \quad (X, X') \in \mathcal{X}, \tag{1}$$

with $\mathrm{P}\{Y \preccurlyeq Y'\} \equiv \mathrm{P}\{Y < Y'\} + \frac{1}{2}\mathrm{P}\{Y = Y'\}$. Here $g(\cdot)$ denotes a link function, Z is a covariate vector that depends on the predictors X and X' and $\mathcal{X}$ is the set of predictors for which the model is defined.

The `pim`-package allows fitting a nearly unlimited range of PIMs through extensive customisations.

1. One can manually provide the set of pairs of observation indices for the pseudo-observations (the "poset"), one can use any of the provided functions (`onewayposet` which includes all unique oneway combinations ($(1,2)$, $(1,3)$ and $(2,3)$), `pairwiseposet` which does the same after ordering the data based on the predictors in the model (the lexicographical order restriction) or the default `fullposet` which simply contains all combinations), and one can even write a custom function for it.

   For example, in the presence of 2 predictors, say $X^T = (X_1, X_2)$, the lexicographical order restricted model is defined for $\mathcal{X} = \{(X, X') | X_1 < X_1' \text{ or if } X_1 = X_1' \text{ then } X_2 \leq X_2'\}$, which can be selected straightforwardly by employing the pairwiseposet function.

2. The link function in the current implementation is restricted to `"identity"`, `"logit"` and `"probit"`. However, through customisation of the estimators (in particular by providing custom implementations of `scorefunctioncreator.default` and `Uforposandwich.default`), this can be easily overcome.

3. By default, the left hand side of the formula (e.g. y in $y \sim x$) is always used for a true probabilistic index $\mathrm{P}\{y \preccurlyeq y'\}$, but $\mathrm{P}\{y \leq y'\}$ and $\mathrm{P}\{y < y'\}$ can also be attained through parameter `lhs`.

4. In the presence of categorical predictors transitivity is assumed.

The function `pim` allows fitting PIMs for different choices of Z. A natural choice is the difference in predictors, i.e. $Z = X' - X$. Unless its parameter `interpretation` is set to `"marginal"`, all predictors (e.g. X_1) that occur in the model formula without altering functions (see below) are indeed interpreted as $X_1' - X_1$. For interactions to also behave as the difference, an extra parameter `interactions.difference` is provided. The defaults are chosen in such a way that the design matrix is created as the difference between the left and right design matrix, but with an intercept added. If you want to avoid the intercept, you have to exclude it from the model as you would in normal formulas, by adding -1 to it.

As an example, model formula $y \sim a * b$ will (by default) represent $\mathrm{P}\{y \preccurlyeq y'\} = \beta_0 + \beta_1(a' - a) + \beta_2(b' - b) + \beta_3(a'b' - ab)$.

Note that the above interpretation (i.e. with parameter `interpretation` equal to `"regular"`) of the model formula will only interpret individual columns and interactions

as differences. If you want to completely enforce the design matrix to be the difference of the design matrices, you can use the default `interpretation="difference"`, which will indeed enforce this. The altering functions are not allowed in this case, and `interactions.difference` is ignored.

Note that when the model satisfies $m(X, X'; \beta) + m(X', X; \beta) = 1$ the lexicographical order restriction corresponds to the NO order restriction and hence the model is defined for all couples of predictors (X, X'), see Thas et al. (2012) for more details.

For expressing more complex models, 4 altering functions are provided: $L(X)$, $R(X)$, $O(X)$ and $F(X)$. These expand to:

1. $L(X)$: the X value of only the left part of the pseudo-observation (with the default suffixes provided, this will be denoted further as X_L)
2. $R(X)$: the X value of only the right part of the pseudo-observation (with the default suffixes provided, this will be denoted further as X_R)
3. $O(X)$: (can only be used on orderable predictors) $I\left(L(X) \preccurlyeq R(X)\right)$
4. $F(X)$: (can only be used on factors) holds all interaction terms where the left value is smaller than the right one.

Finally, when *force.marginal* is *TRUE*, terms X without altering functions are interpreted as $R(X)$. This is typically only useful for marginal models, as specified in TODO rankpaper. Note that some of the altering functions are not relevant in marginal models, and the fit will fail if you try to do so.

In the following sections we illustrate the `pim()` function according to different choices of Z. In Sections 3-5 case studies from Section 6 in Thas et al. (2012) are analyzed, while Section 6 considers categorical predictors. Section 7 gives some conclusions and remarks.

2 pim parameters and their influence

As can be observed in the help for the `pim` function (`?pim`), a lot of parameters have been provided. Some of these require a function passed in themselves, so a large amount of customisation is possible. This section describes in some more detail the effect of each of the parameters.

2.1 Standard parameters

The `pim` function currently supports only the case where a `data.frame`-like object can be passed through `data`, and the model can be expressed as a `formula`. If you want to manually provide the design matrix, you need to create an object of class `pimfitdata` yourself (see `?pim.fit.prep`) and pass this to `pim.fit`.

The other more traditional parameters for a model fitting function are `link` (providing one of the link functions - note that although the provided options suggest otherwise,

currently only "logit", "identity" and "probit" are provided, where "logit" is the default), and na.action (note again that no effort has been made to handle missing data in the fitting code, so there will be little to no use in letting NA values pass. Also: the na.action is applied to the original data, not to the design matrix, so it is typically the strictest possible.)

We have opted to allow for blocking variables not through constructs in the formula, but by simply providing a character vector blocking.variables. Note that these are only used to filter the poset: only combinations of observations that share the same values for the blocking variables are allowed.

Finally, we provide a parameter verbosity: most of the functions within this package supprt it, and it is typically passed on down the stack of functions while diminishing it. At some points, when this variable is above a threshold (typically, above zero), some diagnostic or progress text is displayed. For long running fits, this can be interesting to follow the progress (and then the overhead of the continuous logging will also be relatively small). It can also be used to investigate unexpected results, as it may display intermediate results, so the cause for the unexpected result can typically be pinpointed more quickly. Please be aware that the verbosity comes at a performance cost, so for typical use it is best to leave verbosity at its default of zero.

2.2 Design parameters

There are quite a few parameters that govern how the formula is interpreted to create the design matrix.

2.2.1 poset

This parameter represents either a matrix (or similar structure, though experience shows that matrix is indeed the fastest performing) with two columns holding the rownumbers of the left and right observation in each pseudo-observation (in Thas et al. (2012), this is denoted with a calligraphic I.), or a function that can create this based on the data.

We expect that the option to immediately pass in a matrix will be seldom used.

If you pass a function, it should have three parameters: data, formula and verbosity. The values it will receive are always the ones that were passed along to pim. The function should return a list with two items: data (which should hold either the original data or some reordered or transformed version of it) and poset, which should be the resulting matrix of rownumber combinations. Some functions have already been provided for use in this way, which will be able to handle most cases:

- fullposet: the default option, which will simply create a matrix with all combinations of rownumbers.
- onewayposet: this will contain only combinations where the left rownumber is smaller than the right one

- `pairwiseposet`: will first try to order the data according to the variables in the model (note: if more than one variables is present, the order of the variables is the one in the dataset), and then again will return only combinations where the left rownumber is smaller than the right one. In Thas et al. (2012), this is known as the lexicographical order.
- `forcedcolorderonewayposet(columnnames)`: here, the data is first ordered according to the `columnnames` passed in, and then again returns only combinations where the left rownumber is smaller than the right one.
- `oldpimposet` and `oldpimposetbft` are provided for backward compatibility and should not be used.

2.2.2 `leftsuffix,rightsuffix`

During the creation of the design matrix, it often happens that two versions of the same variable have to be handled, pertaining to the left and right observation. Where necessary, these will be referred to with their original variable names with the matching suffixes appended. The defaults (`_L` and `_R`) will probably suffice for most circumstances (and little to no testing has been done with other suffixes). These parameters are mostly provided for the unlikely event where both columns `X` and `X_L` are already present in the data. Some very basic safety checks are performed in `pimformula` to avoid issues here.

2.2.3 `lhs`

A regular PIM model will always have $\mathrm{P}\{Y \preccurlyeq Y'\}$ as its pseudo-outcome, which will be the interpretation of the left hand side of the `formula` if `lhs="PO"`, the default. For other applications, we also provide $\mathrm{P}\{Y \leq Y'\}$ and $\mathrm{P}\{Y < Y'\}$, by passing `"<="` or `"<"` as `lhs`. Note that in the future, this may be extended (see `pimformula`) to include even more general functions.

2.2.4 `interpretation` and `interactions.difference`

These are probably the most influential parameters. `interactions.difference` is ignored unless `interpretation` is `"regular"`, so the existing combinations are:

- `interpretation="difference"`: In this case, the formula is used to create a `glm` style design matrix of all the original observations (see `model.matrix`). Then the results for the left and right observations are subtracted (though if an intercept was present, this is left out!). It is important to realise that specifying an intercept or not in the `formula` will influence the way the original design matrix is created for factor variables, so consider this carefully. We expect that you will want to include the intercept in most cases, so that the first level of the factor is used as a reference (and gets no dummy variable).

- `interpretation="regular"` and `interactions.difference=TRUE` In this case, main effect terms that are variable names occurring in the `data`, and interaction terms are interpreted as differences between the right and left values of that term, where possible. If no calculated columns are present, this should have the same effect as `interpretation="difference"`, except that intercepts are not excluded. Calculated main effect terms (e.g. `I(X2)`) are interpreted by replacing each column name with the difference (in the example: $(X' - X)^2$), where variables are first converted to numerical (note the impact for factors!!). In addition, the altering functions `L,R,F,O` can be used as shortcuts to some calculated terms (see the end of Section 1)

- `interpretation="regular"` and `interactions.difference=FALSE` This is the same as the previous option, but interaction terms are now treated the same way as calculated terms. The simplest example where the difference is clear, is in the interpretation of $Y \sim A * B$: with `interactions.difference=TRUE`, the interaction term is interpreted as $I((A\_R : B\_R) - (A\_L : B\_L))$, while with `interactions.difference=FALSE`, it is interpreted as $(A\_R - A\_L) : (B\_R - B\_L)$.

- `interpretation="marginal"`: in this interpretation, only the right side value of each variable is allowed in the model. As such, variable names are replaced with their right side value. Some of the altering functions are still allowed, but have slightly different forms (see again at the end of Section 1).

2.3 Fitting parameters

Once the model matrix has been created and the link function is known, fitting the PIM requires estimating the coefficients through solving a set of (potentially nonlinear) equations and then estimating the variance.

For estimating the coefficients, several methods are readily provided, that can be passed as the `estimator` parameter to `pim`:

- `estimator.nleqslv` (the default): will use `nleqslv` to solve the equations, and as such, all parameters that tune its performance can be passed along. In addition, `ignore.error` can be set to `TRUE`, so nonconvergence will be ignored (although we encourage users to solve this through the other parameters to `nleqslv`). The function that calculates the lefthand side of the equations (i.e. what has to be set to zero) based on the `link` function has to be provided as the `scoreFunctionCreator`. Its default, `scorefunctioncreator.default` provides these for `"logit"`, `"identity"` and `"probit"`, but this can be easily extended to other link functions by employing equation 8 of Thas et al. (2012) (see the implementation of `scorefunctioncreator.default` for the requirements of this type of function). Note also that the set of equations for a given link function is uniquely defined, and the current implementation does not claim to hold the most efficient set.

- `estimator.BB`: very similar to `estimator.nleqslv`, although this relies on `BBsolve` from package `BB`, so the parameters reflect this.

- `estimator.glm`: the maximum likelihood estimating equations for the mathing `glm` are of the form of equation 8 in Thas et al. (2012), so any solution to these will be a correct estimate of the PIM coefficients. This is what this estimator provides. A slight disadvantage of this is that the `glm` (co)variance estimate is also calculated although this is not actually usable in most cases (since it requires independence of the pseudo-observations). This can be ignored, but might involve a performance impact for sizeable data.

- `estimator.trymultiple`: this will try all reasonable parameter values for `nleqslv`, and after that `BBSolve` until one works (i.e. does not give an error). This can obviously be very slow, so it is much better to figure out which set of parameters works for a given dataset. This is only intended for lazy people who have plenty of time on their hands.

- `estimator.glmnet`: this estimator is still somewhat in the experimental phase, but will apply elastic net penalization to the estimating equations by employing `glmnet` in a similar manner as `estimator.glm` does with `glm`. The parameters are the natural ones to `glmnet`.

When estimating the sample (co)variances of these parameter estimates, we also have several options. It should be noted that a requirement for the (co)variance estimates to be correct is that the same set of estimating equations is used. This is in no way enforced by `pim`! Because the coefficient estimation occurs completely seperate, and custom functions that represent the equations can be provided, there is no way to keep these in check. This is the responsability of the user of these functions. Note, however that for the provided link functions (`"logit"`, `"identity"` and `"probit"`) and the default estimating functions, this is OK.

The (co)variance estimating options are:

- `varianceestimator.sandwich` (the default): provides a straightforward and highly optimized implementation of the sandwich estimator (theorem 2 in Thas et al. (2012)). Similarly as for `estimator.nleqslv`, a function `Uforposandwich` providing the estimating equations and their partial derivatives has to be procured. Once again, the default for this parameter (`Uforposandwich.default`) provides the matching results for the three provided link functions and `scorefunctioncreator.default`.

- `NULL`: by passing `NULL`, no attempt is made to estimate the (co)variances. This can be useful in e.g. bootstrapping or crossvalidating settings, where these calculations would cause unnecessary overhead.

- `varianceestimator.H0`: when the link function is `"identity"`, a simpler and more efficient estimate of the (co)variances exists when the null hypothesis that all parameters are zero is true. This estimator provides just that estimate.

- `varianceestimator.glm`: can only be used if the coefficient estimation happened through `estimator.glm`: in that case, this estimator returns `glm`'s (co)variance estimate. The user should take care only to use this when the design implies independence of the psuedo-observations.

- `varianceestimator.bootstrap`: the variance can also be estimated by a non-parametric bootstrap (important notice: the bootstrap samples are taken on the original dataset (not simply on the pseudo-observations) to ensure that the covariance pattern is preserved). Besides taking the number of bootstrap iterations (`D`) as a parameter, you can also specify `keep.posetbs=TRUE` (the default being `FALSE`) to also return a list of the resampled pseudo-observations (note: the implementation assumes that all pseudo-observations that can be attained through the poset mechanism on the bootstrap samples, were already present in the orginal poset). You can look at the code of the nonexported function `.basicbootstrap` (`getAnywhere(".basicbootstrap")`) to see how to use this to obtain a bootstrapped `pimfitdata` object from the original one.

2.4 Make-up parameters

Especially with calculated variables, but even when using just the default settings, the names of variables can become quite cluttered, since they now typically involve the differences between the right and left values. Some attempts are taken to make the variable names more readable:

`nicenames`: if this is `TRUE` (the default), the mechanism to make the names more readable is activated. This includes automatic renaming of the differences (with the default suffixes, `X_R-X_L` is then renamed to `X_R-_L`) as well as proper names for the results of the altering functions. `extra.nicenames`: through this dataset, extra variable names (or parts of them) and matching "nicer" names can be passed in to perform the renaming. It is useful to know that the whitespace will be removed from names before attempting a replace. An example on how to use this can be found in `?pim`: see `pimb` there, or also Section 5 below.

2.5 Utility parameters

One final parameter is `keep.data`, which chooses whether or not the design matrix is kept in the final object. We expect that especially for big models, this might be rather big, so the default is not to keep it.

3 Childhood respiratory disease study

For the childhood respiratory disease study we consider the PIM with interaction

$$\begin{aligned}\text{logit}\left(\text{P}\left\{FEV \preccurlyeq FEV'\right\}\right) &= \beta_1(AGE' - AGE) + \beta_2(SMOKE' - SMOKE) \\ &\quad +\beta_3(AGE' * SMOKE' - AGE * SMOKE).\end{aligned}$$

Because this PIM corresponds to a covariate vector Z of the form $Z = X' - X$, with $X^T = (AGE, SMOKE, AGE * SMOKE)$, the formula statement of `pim()` is similar to the formula statement of `lm()` and `glm()`. We first read in the data

```
> library(pim)
> data("FEVData")
> head(FEVData)
```

```
  Age   FEV Height Sex Smoke
1   9 1.708   57.0   0     0
2   8 1.724   67.5   0     0
3   7 1.720   54.5   0     0
4   9 1.558   53.0   1     0
5   9 1.895   57.0   1     0
6   8 2.336   61.0   0     0
```

Here `FEV` stands for the forced expiratory volume (FEV), `Age` for the age of the child (AGE) and `Smoke` whether a child smokes or not ($SMOKE$). We fit the PIM:

```
> library('pim')
> pim.fit1 <- pim(FEV ~ Age*Smoke-1, data = FEVData, link="logit",
+   poset=oldpimposet, estimator=estimator.nleqslv(ignore.error=TRUE),
+   keep.data=TRUE, interpretation="regular")
> pim.fit1
```

```
Call:
pim(formula = FEV ~ Age * Smoke - 1, data = FEVData, link = "logit",
    poset = oldpimposet, interpretation = "regular", estimator = estimator.nleqslv(igno
    keep.data = TRUE)

Coefficients:
      Age_R-_L     Smoke_R-_L Age:Smoke_L-_R
     0.6076003      5.3068852     -0.4553885
```

The estimated model is given by

$$\begin{aligned}\text{logit}\left(\hat{\text{P}}\left\{FEV \preccurlyeq FEV'\right\}\right) &= 0.61(AGE' - AGE) + 5.31(SMOKE' - SMOKE) \\ &\quad -0.46(AGE' * SMOKE' - AGE * SMOKE).\end{aligned}$$

Thus the `lm()`-like formula

~ Age*Smoke = Age + Smoke + Age:Smoke,

is automatically converted to a `pim()`-like formula

~ (Age' - Age) + (Smoke' - Smoke) + (Age':Smoke' - Age:Smoke).

The `summary()` function gives the estimates and corresponding standard errors together with the $Z-$ and p-value corresponding to the null-hypothesis $H_0 : \beta = 0$.

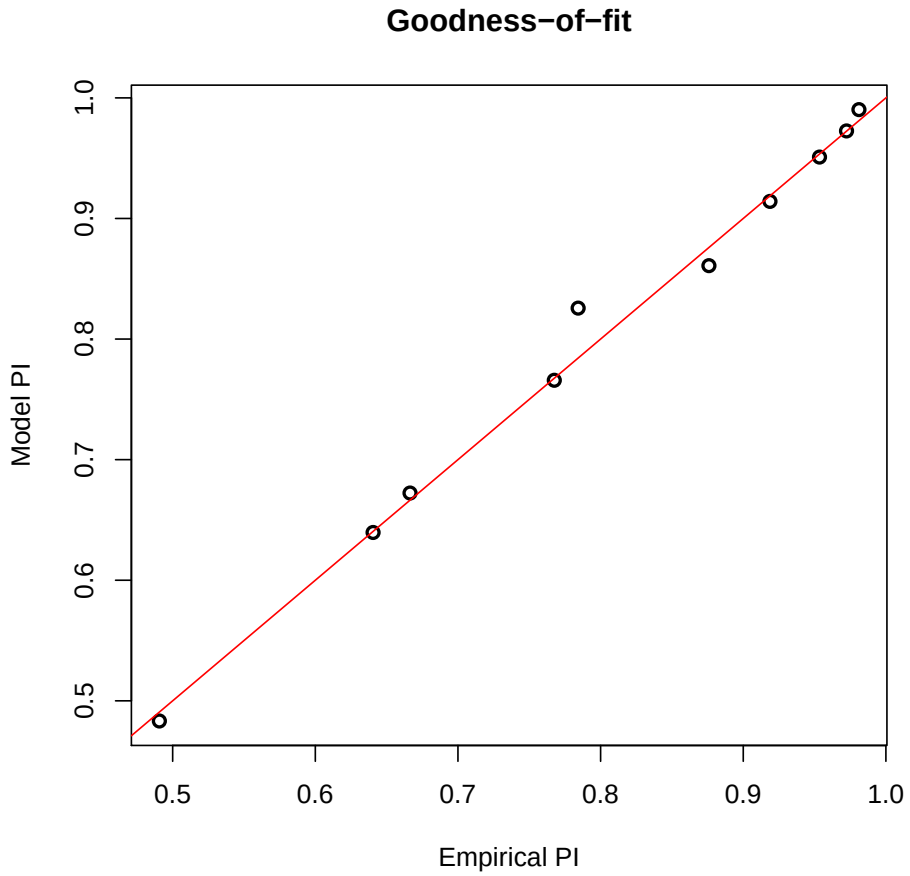
```
> summary(pim.fit1)

Call:
pim(formula = FEV ~ Age * Smoke - 1, data = FEVData, link = "logit",
    poset = oldpimposet, interpretation = "regular", estimator = estimator.nleqslv(igno
    keep.data = TRUE)

                Estimate Std. Error Z value  Pr(>|z|)
Age_R-_L        0.607600   0.030124 20.1697 < 2.2e-16 ***
Smoke_R-_L      5.306885   1.044227  5.0821 3.732e-07 ***
Age:Smoke_L-_R -0.455388   0.078543 -5.7979 6.714e-09 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The `plot()` function provides a rudimentary goodness-of-fit plot.

```
> plot(pim.fit1)
```



The functions `coef()`, `vcov()` and `fitted.values()` provide the estimated coefficients, variance-covariance matrix of $\hat{\beta}$ and the fitted values respectively.

```
> coef(pim.fit1)

      Age_R-_L     Smoke_R-_L Age:Smoke_L-_R
     0.6076003      5.3068852     -0.4553885

> vcov(pim.fit1)

                    Age_R-_L    Smoke_R-_L Age:Smoke_L-_R
Age_R-_L        0.0009074760  0.009485251  -0.0009254122
Smoke_R-_L      0.0094852506  1.090409267  -0.0802054923
Age:Smoke_L-_R -0.0009254122 -0.080205492   0.0061690504

> head(fitted.values(pim.fit1))
```

```
           [,1]
26_222 0.5000000
26_23  0.6473932
222_23 0.6473932
26_59  0.6473932
222_59 0.6473932
23_59  0.5000000
```

We end this section with an illustration of the interpretation of the age effect. For 2 randomly selected children with the same smoking status and a year difference in age, the probability that the eldest has a higher FEV is estimated by expit$(0.61-0.46SMOKE)$. For non-smokers this probability is 0.65, while for smokers this becomes 0.54.

4 Mental health study

For the mental health study the following PIM was proposed

$$\text{logit}\left(\text{P}\left\{MI \preccurlyeq MI'\right\}\right) = \beta_1(SES' - SES) + \beta_2(LI' - LI). \tag{2}$$

```
> data("MHData")
> head(MHData)
```

```
  mental ses life
1      1   1    1
2      1   1    9
3      1   1    4
4      1   1    3
5      1   0    2
6      1   1    0
```

Here `mental` stands for the mental impairment (MI), `ses` for the socioeconomic status (SES) and `life` for the life index (LI). Similar as in the previous example we can specify a `lm()`-like formula.

```
> pim.fit2a <- pim(mental ~ ses + life -1, data = MHData, link="logit",
+   poset=oldpimposet, estimator=estimator.nleqslv(ignore.error=TRUE),
+   keep.data=TRUE, interpretation="regular")
> summary(pim.fit2a)
```

```
Call:
pim(formula = mental ~ ses + life - 1, data = MHData, link = "logit",
    poset = oldpimposet, interpretation = "regular", estimator = estimator.nleqslv(igno
    keep.data = TRUE)
```

```
           Estimate Std. Error Z value Pr(>|z|)
ses_R-_L  -0.740163   0.343575 -2.1543 0.031217 *
life_R-_L  0.201179   0.073371  2.7419 0.006108 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The model is estimated by

$$\text{logit}\left(\hat{\text{P}}\left\{MI \preccurlyeq MI'\right\}\right) = -0.74(SES' - SES) + 0.2(LI' - LI).$$

Model (2) can be extended as follows

$$\text{logit}\left(\text{P}\left\{MI \preccurlyeq MI'\right\}\right) = \beta_1(SES' - SES) + \beta_2(LI' - LI) + \beta_3 SES + \beta_4 LI.$$

If we want to fit this model, we need to specify the `formula` statement explicitly because Z is no longer of the form $Z = X' - X$. Some notation is needed to specify the predictors corresponding to the left response in $\text{P}\left\{MI \preccurlyeq MI'\right\}$, thus (SES, LI) and the predictors corresponding to the right response, thus (SES', LI'). The altering functions can be used for this: `L()` for the predictors corresponding to the left response and `R()` for the predictors corresponding to the right response. Thus (SES, LI) in R becomes `(L(ses), L(life)` and (SES', LI') becomes `(R(ses), R(life)`. The `I()` statement is needed to specify specific functions. The function

$$\beta_1(SES' - SES) + \beta_2(LI' - LI) + \beta_3 SES + \beta_4 LI,$$

in R becomes

```
 ~ ses + life + L(ses) + L(life) - 1
```

```
> pim.fit2b <- pim(mental ~ ses + life + L(ses) + L(life) - 1, data = MHData,
+   link="logit", poset=oldpimposetbft,
+   estimator=estimator.nleqslv(ignore.error=TRUE),keep.data=TRUE,
+   interpretation="regular")
> summary(pim.fit2b)

Call:
pim(formula = mental ~ ses + life + L(ses) + L(life) - 1, data = MHData,
    link = "logit", poset = oldpimposetbft, interpretation = "regular",
    estimator = estimator.nleqslv(ignore.error = TRUE), keep.data = TRUE)

           Estimate Std. Error Z value Pr(>|z|)
ses_R-_L  -0.670723   0.382665 -1.7528 0.079642 .
life_R-_L  0.205459   0.069989  2.9356 0.003329 **
```

```
ses_L     -0.034676   0.163157 -0.2125 0.831693
life_L    -0.021601   0.039843 -0.5422 0.587711
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The estimated model is given by

$$\text{logit}\left(\hat{\text{P}}\left\{MI \preccurlyeq MI'\right\}\right) = -0.67(SES' - SES) + 0.21(LI' - LI) - 0.03SES - 0.02LI.$$

5 Food expenditure study

Because of heteroscedasticity the following PIM is proposed to analyze the food expenditure data.

$$\text{logit}\left(\text{P}\left\{FE \preccurlyeq FE'\right\}\right) = \beta\frac{HI' - HI}{\sqrt{HI' + HI}}.$$

```
> data("Engeldata")
```

Here `income` denotes the household income (HI) while `foodexp` denotes the food expenditure (FE). The covariate vector Z is not of the form $Z = X' - X$, hence we need to specify the formula explicitly.

```
> pim.fit3 <- pim(foodexp ~ I((R(income)-L(income))/sqrt(R(income)+L(income)))-1,
+   data = Engeldata, link="logit", poset=oldpimposetbft,
+   estimator=estimator.nleqslv(ignore.error=TRUE),
+   keep.data=TRUE, interpretation="regular",
+   extra.nicenames=data.frame(
+     org="I((R(income)-L(income))/sqrt(R(income)+L(income)))",
+     nice="weightedincomediff", stringsAsFactors=FALSE))
> summary(pim.fit3)

Call:
pim(formula = foodexp ~ I((R(income) - L(income))/sqrt(R(income) +
    L(income))) - 1, data = Engeldata, link = "logit", poset = oldpimposetbft,
    interpretation = "regular", estimator = estimator.nleqslv(ignore.error = TRUE),
    keep.data = TRUE, extra.nicenames = data.frame(org = "I((R(income)-L(income))/sqrt(
        nice = "weightedincomediff", stringsAsFactors = FALSE))

                   Estimate Std. Error Z value  Pr(>|z|)
weightedincomediff  0.38971    0.02436  15.998 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The estimated model is given by

$$\text{logit}\left(\hat{\text{P}}\left\{FE \preccurlyeq FE'\right\}\right) = 0.39\frac{HI' - HI}{\sqrt{HI' + HI}}.$$

6 Categorical predictors

In the presence of categorical predictors, a dummy coding is used together with the covariate vector $Z = X'_{dummy} - X_{dummy}$, where X_{dummy} denotes the dummy coding of the predictor X. This model is inspired by the relation between a linear models and a PIM. As an example consider a predictor X with 3 levels A, B and C with dummy coding $X_B = 1$ if $X = B$ and $X_B = 0$ otherwise and $X_C = 1$ if $X = C$ and $X_C = 0$ otherwise. The linear model

$$Y = \alpha_0 + \alpha_1 X_B + \alpha_2 X_C + \varepsilon,$$

with $\varepsilon \sim \mathrm{N}(0, \sigma^2)$ embeds the PIM

$$\mathrm{P}\left\{Y \preccurlyeq Y' | X, X'\right\} = \Phi\left\{\beta_1 (X'_B - X_B) + \beta_2 (X'_C - X_C)\right\},$$

where $\beta_i = \alpha_i / \sqrt{2\sigma^2}$. Note that the PIM has only 2 parameters (β_1 and β_2) to model 3 probabilities $\mathrm{P}\left\{Y \preccurlyeq Y' | X = A, X' = B\right\}$, $\mathrm{P}\left\{Y \preccurlyeq Y' | X = A, X' = C\right\}$ and $\mathrm{P}\left\{Y \preccurlyeq Y' | X = B, X' = C\right\}$. This is a consequence of the transitivity assumption which is implied by the linear model:

$$\begin{aligned} \mathrm{P}\left\{Y \preccurlyeq Y' | X = B, X' = C\right\} &= \Phi\left\{\Phi^{-1}\left(\mathrm{P}\left\{Y \preccurlyeq Y' | X = A, X' = C\right\}\right)\right. \\ &\quad \left. -\Phi^{-1}\left(\mathrm{P}\left\{Y \preccurlyeq Y' | X = A, X' = B\right\}\right)\right\}. \end{aligned}$$

In R this becomes

```
> n <- 100
> X <- factor(sample(LETTERS[1:3], n, replace = TRUE))
> Y <- model.matrix(~ X)%*%c(1,2,3) + rnorm(n)
> data.tmp <- data.frame(Y, X)
> pim.fit4 <- pim(Y ~ X-1, data = data.tmp, link = "probit", poset=oldpimposet,
+   estimator=estimator.nleqslv(ignore.error=TRUE), keep.data=TRUE,
+   interpretation="regular")
> summary(pim.fit4)

Call:
pim(formula = Y ~ X - 1, data = data.tmp, link = "probit", poset = oldpimposet,
    interpretation = "regular", estimator = estimator.nleqslv(ignore.error = TRUE),
    keep.data = TRUE)

         Estimate Std. Error Z value  Pr(>|z|)
X_R-_L_B  1.22222    0.23016  5.3102 1.095e-07 ***
X_R-_L_C  1.97852    0.27016  7.3235 2.416e-13 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The estimated model is given by

$$\Phi^{-1}\left(\hat{\mathrm{P}}\left\{Y \preccurlyeq Y' | X, X'\right\}\right) = 1.22(X'_B - X_B) + 1.98(X'_C - X_C).$$

Note that PIMs are semiparametric, thus the normality assumption is not required to obtain consistent and asymptotically normally distributed estimators. The linear model merely serves as a guide on how Z can be constructed based on the predictors X and X'.

7 Conclusions and remarks

The `pim`-package is illustrated on several examples and allows fitting a broad class of PIMs. PIMs which are embedded by a linear model as well as less restrictive PIMs are allowed. For categorical predictors however, only PIMs which are based on a linear model can be constructed.

Note that for a sample size of n a total $n(n-1)/2$ pseudo-observations are created. Consequently for large sample sizes the function goes quite slow.

In one of the next versions the above mentioned shortcomings will be tackled. All bugs/comments/suggestions are welcome at JanR.DeNeve@Ugent.be.

References

O. Thas, J. De Neve, L. Clement, and J-P. Ottoy. Probabilistic index models (with discussion). *Journal of the Royal Statistical Society - Series B*, 74:1–29, 2012.