PLS-PM Implementation Report

Dimitri Chikhladze, Jonathan Janke, Josefine Kuka

08.06.2017

Run preferences

The run preferences can be defined in the Main-part of our code, which is in the end of the R-file

- **input data filename**, e.g. "bank.csv"
- **inner model mapping filename**, e.g. "inner_model_mappings.txt" with LVs as source and target
- **outer outer model mapping filename**, e.g. "outer_model_mappings.txt" with LVs and MVs as source or target
- **imputation method** (imputeDeletion, imputeMean, imputeNearestNeighbour or imputeSpecialMean)
- **inner approximation method** (pathWeighting, centroidWeighting or factorialWeighting)
- **threshold for PLS termination**, e.g. 0,000001
- **bootstrap sample number**, e.g. 500

The inner and outer model is specified in a comma-seperated file with 2 columns indicating the source and the target. The input files are read and will determine the mode of each block.

Data Preperation

After reading the specified input files, the data is cleaned and missing valued imputed (**dataTreatment(raw_data, impute_method)**). The user can choose between 4 different imputation methods, which will have a small effect on the end results:

- imputation by deletion (imputeDeletion): Delete all rows containing missing values
- impuation with mean (imputeMean): mean value in the row (mean of ratings of the same person)
- imputation with k-nearest neighbours (imputeNearestNeighbour): k most similar rows are used to predict the missing value
- imputation with special mean (imputeSpecialMean): mean of column + (avg rating in row - avg all ratings)

Furthermore, the data is normalized (mean=0, standard deviation=1) as this is a requirement for the algorithm.

Model Initialization

The initialization is called by **my.pls.model(data, inner_model_mappings, outer_model_mappings)**. All necessary blocks and matrices are created and stored in a list, called **initial_model**:

- Latent Variables (LV)
- Manifest Variables (MV)
- Inner model as binary matrix (inner_model)
- Blocks defining the mode and manifest variables for each LV (blocks)
- outer model weights, initialized binary (outer_weights)

```
initial_model$blocks$Expectations
```

```
## $predecessors
## [1] "Image"
##
```

```
## $successors
## [1] "Quality"      "Value"        "Satisfaction"
##
## $MV
## [1] EXPE1 EXPE2 EXPE3
## 24 Levels: EXPE1 EXPE2 EXPE3 IMAG1 IMAG2 IMAG3 IMAG4 IMAG5 LOYA1 ... VALU2
##
## $mode
## [1] "A"
```

PLS PM

The actual PLS algorithm needs the imputed data, the inital model, the defined inner approximation method and a termination threshold as input parameters: **my.pls(data, initial_model, inner_approximation_method, threshold)**

The my.pls method performs the partial least square path modeling algorithm starting with binary outer model weights. The iterative part estimates the outer weigths and latent variables scores. These are used to calcultate the path coefficents and further measures. The returned list includes:

- weights
- path_coefficients
- total_effects
- crossloadings
- loadings
- scores
- blocks

```
model$path_coefficients
```

```
##              Image Expectations   Quality     Value Satisfaction   Loyalty
## Image            0    0.7019229 0.0000000 0.0000000 -0.009942224 0.1311115
## Expectations     0    0.0000000 0.7596794 0.2887979  0.351876312 0.0000000
## Quality          0    0.0000000 0.0000000 0.5408914  0.374498009 0.0000000
## Value            0    0.0000000 0.0000000 0.0000000  0.190299069 0.0000000
## Satisfaction     0    0.0000000 0.0000000 0.0000000  0.000000000 0.7255517
## Loyalty          0    0.0000000 0.0000000 0.0000000  0.000000000 0.0000000
```

Additional Methods

Besides the standard PLS-PM with several imputation and inner model approximation options, the following enhancements are implemented.

Handling Formative Blocks

If the input is formative or reflective is defined by source and target in the input file of the outer mappings. If the sources are manifest and target are latent variables, the block is formative (Mode B). If the sources are latent and target are manifest variables, the block is reflective (Mode A). The presented code can handle both modes in the same model: all relations in one block have to be of the same mode, but different blocks can have different modes. The blocks can be accessed from the initial model (initial_model) and from the final PLS model.

Assessment Measures

The assessment measures can be calculated after running the PLS algorithm (**assessmentMeasures(data, initial_model, result)**), which will run all assessment measures on the provided PLS model for the latent variables depending on the mode. The input for this function is the normalized data, the initial model and the results gained from running the pls algorithm. Through the variable **assessment** following measures are presented:

structural_model_test:

- R-Squared indexes (r_sqare)
- Redundancy indexes (redundacy)
- Goodness-of-fit index (goodness_of_fit)

reflective:

- Cronbach's alpha (cronbachs_alpha)
- Dillon-Goldstein's rho (dillon_goldstein)
- Communality indexes (communality)
- Average Variance Extracted (communality$average_variance_extracted)
- Eigenvale Analysis (eigenvalue_analysis)
- Insignificant MV (insignificant_manifest_variables_based_on_communality)
- Validation of Cross Loadings (cross_loadings_validation)

formative:

- Collin measure (collin_measure)

```
assessment <- assessmentMeasures(data, initial_model, model)
assessment$reflective$dillon_goldstein$Image
```

```
## [1] 0.9127009
```

```
assessment$reflective$cronbachs_alpha$Image
```

```
## [1] 0.8802463
```

Bootstrapping

The user has the option to perform bootstrapping on the data set (**bootstrap(data, sample_number=500)**). In this case, the algorithm will be run 500 times on random draws from the original data. The result consists of 500 models returned in a list.

The significance of the average path coefficients can be tested by **testSignificance(model_list, significance_level = 0.05)**. The function returns a list containing:

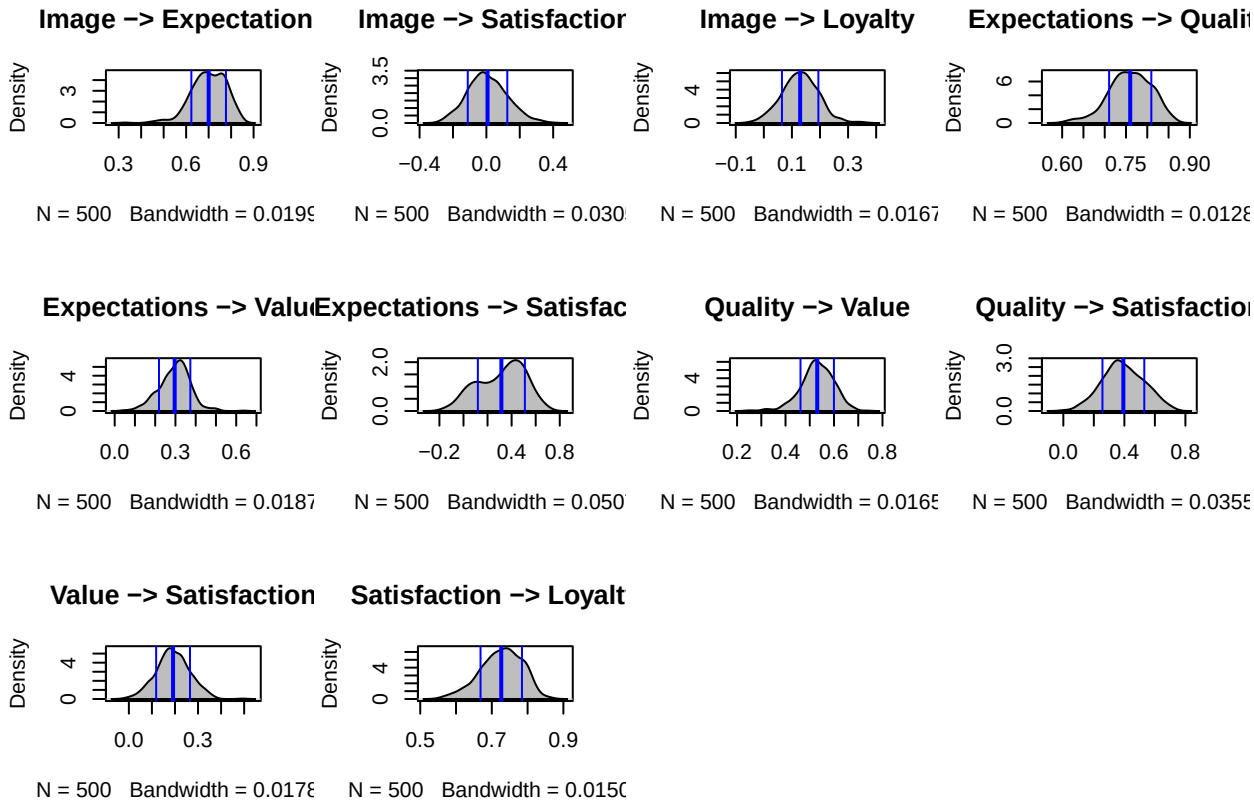
- boolean of significance of inner model connections (significance)
- significance scores (sig_scores)
- critical significance scores (sig_level)

```
testSignificance(bootstrap_model)$significance
```

```
##              Image Expectations Quality Value Satisfaction Loyalty
## Image           NA          TRUE      NA    NA         FALSE    TRUE
## Expectations    NA            NA    TRUE  TRUE         FALSE      NA
## Quality         NA            NA      NA  TRUE          TRUE      NA
## Value           NA            NA      NA    NA          TRUE      NA
## Satisfaction    NA            NA      NA    NA            NA    TRUE
## Loyalty         NA            NA      NA    NA            NA      NA
```

Furthermore, a density plotting was implemented, showing the densitiy of each path coefficent including the mean and standard deviation:

```
invisible(plotBootstrap(bootstrap_model))
```



Consistent PLS

The consistent PLS can be run by **consistentPLS(data, initial_model)**. The method will run the normal PLS first as this is a requirement to proceed with the consistent measures. For each latent variable, the Dijkstra-Henseler-Rho is calculated (using the weights "w" and correlation matrix of the data "S"). Afterwards, the following results are generated and returned:

- Dijkstra-Henseler-Rho (rho)
- Consistent correlation (consist_cor)
- Consistent loadings (consist_loadings)
- consistent path coefficients (consist_path_coeffs)
- consistent r squared (consistent_r_squared)
- consistent mean r_squared (consistent_mean_r_squared)

```
consistent_model <- consistentPLS(data, initial_model)
consistent_model$rhos
```

```
##        Image Expectations      Quality        Value Satisfaction
##    0.8855915    0.8930749    0.9170507    0.8121050    0.8474250
##      Loyalty
##    0.8649182
```