Model Shift and Model Risk Management

The R Package ModelShift

Andrija Djurovic

Dr. Alan Forrest

Contents

1 The R Package ModelShift

The ModelShift package provides a framework for quantifying model shifts in logistic regression with categorical risk factors, one of the most commonly used methods in Probability of Default (PD) modeling.

The development version is currently available on the following GitHub page.

To install the ModelShift package, practitioners can run the following command:

```
devtools::install_github(repo = "andrija-djurovic/adsfcr",
                         subdir = "mrm/ms_r",
                         upgrade = "never")
```

The core of the package is the ms.pd function, which accepts the following arguments:

- db: A data frame containing the modeling dataset.
- target: A vector specifying the name of the target (dependent) variable.
- rf: A vector of risk factor names.
- data.shift: A data frame with column names from rf and the last column named "n".
- encoding: The encoding method, with available options: "dummy" and "WoE".
- method: The model shift method, with available options: "mm" (matrix multiplication), "wbr" (weighted binomial regression), and "wfr" (weighted fractional regression).
- woe.tbl: The manually created WoE table. A data frame with three columns: "rf", "bin", and "woe". The default value is NULL..
- lr.i: The glm object of the initial model. The default is NULL. If supplied, the function uses the given model instead of estimating it.

Before presenting the simulation study and demonstrating the functionalities of the `ModelShift` package, the following sections provide a brief overview of the model shift concept, its applications, and different methods for its quantification.

2 Model Shift

Model shift refers to the quantitative change in a model's parameters and outputs resulting from shifts in the input data. It is particularly useful in credit risk modeling for understanding how models react to changes in their underlying assumptions, data, or environment. The concept of model shift enables practitioners to:

- efficiently address "What if...?" scenarios, quantifying how data shifts impact model outputs without needing complete model redevelopment;
- provide a systematic way to assess and respond to model sensitivities and weaknesses, enhancing model validation, monitoring, and risk management.

In this framework both data and models can be presented as observed and expected counts in a high-dimensional contingency table. These in turn are converted to points in a data space of high dimensional vectors. Here a data point is defined by the proportion of the observed population in each cell, viewed as a vector of real numbers indexed by the cells. Likewise, the model point is defined by the proportion of the expected population in each cell etc. We can keep track separately of the total population size for purposes of inference, but note that the data point and model point do not vary as population size changes. The maximum likelihood construction of logistic regression model from data depends solely on the proportions.

3 Applications of Model Shift

Model shift in response to data shift is more than an academic exercise. It is clearly closely related to data drift and concept drift, well known concerns in model management, and it underlies modeling techniques such as imputation, bootstrapping and rebalancing. We are interested in it as a method for important analyses in modern model risk management.

The following list outlines some use cases of the model shift:

- *Dynamic Model Reweighting*: Enables real-time updates of model parameters as new data streams in, ensuring agility in model adjustments without waiting for periodic reviews.
- *Prioritizing Validation Investigations*: Quickly computes model shifts for various data shift scenarios, enabling efficient triage and focus on the most impactful concerns.
- *Quantifying Business Impacts*: Links data shifts to business-relevant metrics like Probability of Default (PD) in Risk-Weighted Assets (RWA), ensuring sensitivity analyses are connected to actionable outcomes.
- *Sensitive Data Shift Identification*: Enables the identification of data shifts that have the most significant impact on models, enriching the validation narrative with actionable insights.
- *Bespoke Model Monitoring*: Defines monitoring metrics for sensitive data shifts, creating early warning systems, particularly for population shifts that do not immediately affect model outputs.
- *Automated Validation and Monitoring*: Streamlines validation and monitoring processes, integrating them with dynamic model updates for continuous, real-time risk management.

4 Methods for Quantifying Model Shift

The methods for quantifying model shift are:

1. matrix multiplication approach;
2. weighted binomial logistic regression;
3. weighted quasi-binomial regression (weighted fractional logistic regression).

Dr. Alan Forrest proposes a first-order approximation using a matrix multiplication approach to quantify changes in model parameters directly.

Andrija Djurovic introduces two exact alternative methods (weighted binomial and fractional logistic regression) based on re-estimating model parameters, both of which can address the same task.

All three approaches are related to the widely used binomial logistic regression method with categorized risk factors commonly employed in developing PD models.

4.1 Matrix Multiplication Approach

The first-order model shift (Δp) can be explicitly represented as a matrix multiplication of the data shift. The following formulas illustrate the process of approximating parameter changes given the data shifts (Δx^+ and Δx^-):

$$\Delta p = C^{-1} D^T \left[(I + Z)^{-1} \Delta x^+ - (I + Z^{-1})^{-1} \Delta x^- \right]$$

where:

- D is the design matrix;
- Y^+ and Y^- are the diagonal matrices of modeled frequencies restricted to binary output 1 and 0, respectively;
- $Z = Y^+(Y^-)^{-1}$ is diagonal matrix of modeled odds ratios;
- I is identity matrix dimensions `nrow(Z) x nrow(Z)`;
- $Y = (I + Z)^{-1}(I + Z^{-1})^{-1}(Y^+ + Y^-)$;
- $C = D^T Y D$;
- Δx^+ and Δx^- are the shifts in the proportions of input factors for binary outputs 1 and 0, respectively.

4.2 Weighted Binomial Logistic Regression

Another way to quantify changes in the parameters of the logistic regression based on the data shift is to re-estimate the weighted binomial logistic regression.

The following formula presents the log-likelihood function used to estimate the parameters ($\boldsymbol{\beta}$) of the weighted logistic regression::

$$\mathcal{L}(\beta) = \sum_{i=1}^{n} w_i \left[y_i \log \left(\frac{1}{1 + \exp(-\mathbf{x}_i \boldsymbol{\beta})} \right) + (1 - y_i) \log \left(1 - \frac{1}{1 + \exp(-\mathbf{x}_i \boldsymbol{\beta})} \right) \right]$$

where:

- y_i is the binary response variable for the `i`-th observation (either 0 or 1);
- $\mathbf{x}_i$ is the vector of predictors for the `i`-th observation;
- w_i is the associated weight of the `i`-th observation.

4.3 Weighted Quasi-Binomial Regression

The third method for quantifying changes in logistic regression parameters, based on the data shift, is by re-estimating the weighted quasi-binomial regression. Unlike binomial logistic regression, which requires a dichotomous target (0/1), weighted quasi-binomial regression processes fractions between 0 and 1. The weighted fractional logistic regression parameters can be estimated similarly to binomial logistic regression by maximizing the log-likelihood function with an additional term to account for dispersion. Since the additional term affects only the standard error of estimates, the estimated coefficients between weighted binomial and weighted quasi-binomial regression are identical.

The following formula presents the log-likelihood function used to estimate the model parameters ($\boldsymbol{\beta}$), along with the adjustment of the variance-covariance matrix ($\hat{\Sigma}$) based on the dispersion parameter:

$$\mathcal{L}(\beta) = \sum_{i=1}^{n} w_i \left[y_i \log \left(\frac{1}{1 + \exp(-\mathbf{x}_i \boldsymbol{\beta})} \right) + (1 - y_i) \log \left(1 - \frac{1}{1 + \exp(-\mathbf{x}_i \boldsymbol{\beta})} \right) \right]$$

$$\hat{\Sigma} = \hat{\Phi}\hat{V}$$

where:

- y_i is the binary response variable for the `i`-th observation (either 0 or 1);
- $\mathbf{x}_i$ is the vector of predictors for the `i`-th observation;
- w_i is the associated weight of the `i`-th observation;
- $\hat{\Phi}$ is the estimate of the dispersion parameter;
- $\hat{V}$ is the estimated variance-covariance matrix assuming a binomial distribution (the "naive" variance-covariance matrix).

5 Simulation Study

The following steps outline the simulation framework used to quantify changes in model parameters based on a simulated scenario:

1. Assume a simplified PD model consisting of the target variable `Creditability` and two categorical risk factors: `Account_Balance` and `Maturity`. The following `R` code loads the `ModelShift` package, imports the simulation dataset, and checks the dataset structure:

```
suppressMessages(library(ModelShift))
fp <- "https://raw.githubusercontent.com/andrija-djurovic/adsfcr/main/mrm/ms_db.csv"
db <- read.csv(file = fp,
               colClasses = c("Creditability" = "numeric",
                              "Account_Balance" = "character",
                              "Maturity" = "character"),
               header = TRUE)
  str(db)
```

```
## 'data.frame':    1000 obs. of  3 variables:
##  $ Creditability  : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Account_Balance: chr  "01" "01" "02" "01" ...
##  $ Maturity       : chr  "03 [16,36)" "02 [8,16)" "02 [8,16)" "02 [8,16)" ...
```

2. The risk factor `Account_Balance` includes four categories with the following distribution of observations:

```
table(db$Account_Balance)
```

```
##
##  01  02  03  04
## 274 269  63 394
```

3. The risk factor `Maturity` includes five categories with the following distribution of observations:

```
table(db$Maturity)
```

```
##
## 01 (-Inf,8)   02 [8,16)  03 [16,36)  04 [36,45) 05 [45,Inf)
##          87         344         399         100          70
```

4. The final PD model is estimated using binomial logistic regression and dummy encoding in the form: `Creditability ~ Account_Balance + Maturity` with the following estimated coefficients:

```
glm(formula = Creditability ~ Account_Balance + Maturity,
    family = "binomial",
    data = db)
```

```
##
```

```
## Call:  glm(formula = Creditability ~ Account_Balance + Maturity, family = "binomial",
##     data = db)
##
## Coefficients:
##         (Intercept)    Account_Balance02    Account_Balance03    Account_Balance04
##             -1.3234              -0.5064              -1.0873              -2.0194
##   Maturity02 [8,16)   Maturity03 [16,36)   Maturity04 [36,45)  Maturity05 [45,Inf)
##              0.9783               1.4282               1.8817               2.4041
##
## Degrees of Freedom: 999 Total (i.e. Null);  992 Residual
## Null Deviance:       1222
## Residual Deviance: 1042  AIC: 1058
```

5. Assume the following scenario: the portfolio structure changes as the bank plans to increase loan approvals for riskier groups, specifically clients in the `Account_Balance` category `01`, by 40%. Simultaneously, loan approvals for clients in category `04` will decrease by the same number. Given this scenario, the new allocation of `Account_Balance` modalities is:

```
#new count for Account_Balance modality 01
ab_i <- sum(db$Account_Balance%in%"01")*(1 + 0.40)
#new count for Account_Balance modality 04
ab_d <- sum(db$Account_Balance%in%"04") - (ab_i - sum(db$Account_Balance%in%"01"))
#data shift
data.shift <- data.frame("Account_Balance" = names(table(db$Account_Balance)),
                         n = c(ab_i, table(db$Account_Balance)[2:3], ab_d),
                         check.names = FALSE)
#print data.shift
data.shift
```

```
##   Account_Balance     n
## 1              01 383.6
## 2              02 269.0
## 3              03  63.0
## 4              04 284.4
```

Note: The function `ms.pd` accepts a data frame for the `data.shift` argument, which contains the names of some or all risk factors used to estimate the final model. The last column must be named `"n"`.

6. Based on this scenario and the resulting portfolio structure changes, the objective is to quantify the change in the estimated parameters of the final PD model using the three methods available in the `ms.pd` function.

Let's start with the matrix multiplication method to demonstrate how model shift can be quantified using the `ms.pd` function, following the dummy encoding presented in the simulation design above. For this method, the function outputs a list of two elements: the first element (`ms`) quantifies the model parameter shift, while the second element (`db.s`) presents a summary table used for calculating the inputs for the matrix multiplication method.

```
res.mm <- ms.pd(db = db,
                target = "Creditability",
                rf = c("Account_Balance", "Maturity"),
                data.shift = data.shift,
                encoding = "dummy",
                method =  "mm")
#output names
names(res.mm)
```

```
## [1] "ms"   "db.s"
```

```
#parameters shift
res.mm[["ms"]]
```

```
##         (Intercept)   Account_Balance02   Account_Balance03   Account_Balance04
##         0.014977152         0.005604455        -0.005485861         0.004087598
##   Maturity02 [8,16)  Maturity03 [16,36)  Maturity04 [36,45) Maturity05 [45,Inf)
##        -0.002716596        -0.015953224        -0.015774832        -0.092029407
```

To compare the results of the approximate matrix multiplication method with the exact methods (`"wbr"`, `"wfr"`) for the same simulation design, practitioners can run the code below. Unlike the `"mm"` method, these two methods return a list of four elements: the initial `glm` model object (`lr.i`), the simulated `glm` model object (`lr.s`), the model parameter shift (`ms`), and the summary table (`db.s`).

1. Weighted Binomial Logistic Regression:

```
res.wb <- ms.pd(db = db,
                target = "Creditability",
                rf = c("Account_Balance", "Maturity"),
                data.shift = data.shift,
                encoding = "dummy",
                method =  "wbr")
#output names
names(res.wb)
```

```
## [1] "lr.i" "lr.s" "ms"   "db.s"
```

```
#parameters shift
res.wb[["ms"]]
```

```
##         (Intercept)   Account_Balance02   Account_Balance03   Account_Balance04
##         0.015796567         0.005588795        -0.005248640         0.004220945
##   Maturity02 [8,16)  Maturity03 [16,36)  Maturity04 [36,45) Maturity05 [45,Inf)
##        -0.004842827        -0.016517199        -0.015032368        -0.092309301
```

2. Weighted Quasi-Binomial Regression (Weighted Fractional Logistic Regression):

```
res.wf <- ms.pd(db = db,
                target = "Creditability",
                rf = c("Account_Balance", "Maturity"),
                data.shift = data.shift,
                encoding = "dummy",
                method =  "wfr")
#output names
names(res.wf)
```

```
## [1] "lr.i" "lr.s" "ms"   "db.s"
```

```
#parameters shift
res.wf[["ms"]]
```

```
##         (Intercept)   Account_Balance02   Account_Balance03   Account_Balance04
##         0.015796567         0.005588795        -0.005248640         0.004220945
##   Maturity02 [8,16)  Maturity03 [16,36)  Maturity04 [36,45) Maturity05 [45,Inf)
##        -0.004842827        -0.016517199        -0.015032368        -0.092309301
```

As can be seen from the above results, the first-order approximation - matrix multiplication method - produces results comparable to those of the exact methods.

The previous simulation demonstrated an example of dummy encoding, while a more common approach is Weights of Evidence (`WoE`) encoding for risk factors. To run the same exercise using WoE encoding,

practitioners can change the `encoding` argument to `"WoE"` and compare the results. The following R code uses a different encoding method to compare the parameter shift results for the same data shift.

1. Matrix Multiplication with WoE Encoding:

```
res.mm <- ms.pd(db = db,
                target = "Creditability",
                rf = c("Account_Balance", "Maturity"),
                data.shift = data.shift,
                encoding = "WoE",
                method =  "mm")
```

2. Weighted Quasi-Binomial Regression with WoE Encoding:

```
res.wf <- ms.pd(db = db,
                target = "Creditability",
                rf = c("Account_Balance", "Maturity"),
                data.shift = data.shift,
                encoding = "WoE",
                method =  "wfr")
```

The following data frame compares the results from the above runs.

```
data.frame(mm.woe = res.mm$ms,
           wf.woe = res.wf$ms)
```

```
##                           mm.woe       wf.woe
## (Intercept)         -0.003060810 -0.002458801
## Account_Balance.woe  0.006197485  0.005042942
## Maturity.woe         0.037268152  0.035792762
```

The previous simulation runs assumed that WoE is calculated based on the observed data supplied in the `db` argument. However, practitioners often adjust WoE values during model development for specific bins or modalities of risk factors. The most common example involves adjustments for bins containing missing or special values.

The `ms.pd` function accepts a manually supplied WoE table to accommodate this. This table, provided via the `woe.tbl` argument, undergoes specific checks, so practitioners should prepare it as a data frame with three columns: `"rf"`, `"bin"`, and `"woe"`.

To demonstrate this feature of the `ms.pd` function, we first define a function to calculate WoE, apply it to the analyzed risk factors `Account_Balance` and `Maturity`, and then manually adjust one of the observed WoE values.

```
#woe function
woe.calc <- function(db, x, y) {
      tbl.s <- db %>%
               group_by_at(c("bin" = x)) %>%
               summarise(no = n(),
                         ng = sum(1 - !!sym(y), na.rm = TRUE),
                         nb = sum(!!sym(y), na.rm = TRUE)) %>%
               ungroup() %>%
               mutate(pct.o = no / sum(no, na.rm = TRUE),
                      pct.g = ng / sum(ng, na.rm = TRUE),
                      pct.b = nb / sum(nb, na.rm = TRUE),
                      dr = nb / no,
                      so = sum(no, na.rm = TRUE),
                      sg = sum(ng, na.rm = TRUE),
                      sb = sum(nb, na.rm = TRUE),
```

```
                      dist.g = ng / sg,
                      dist.b = nb / sb,
                      woe = log(dist.g / dist.b),
                      iv.b = (dist.g - dist.b) * woe,
                      iv.s = sum(iv.b)) %>%
               as.data.frame()
      woe.v <- tbl.s$woe
      names(woe.v) <- tbl.s$bin
      woe.trans <- unname(woe.v[db[, x]])
return(list(summary.tbl = tbl.s, x.trans = woe.trans))
}
#risk factors encoding
rf <- c("Account_Balance", "Maturity" )
rf.l <- length(rf)
woe.tbl <- vector("list", rf.l)
for   (i in 1:rf.l) {
      rf.i <- rf[i]
      woe.tbl.i <- woe.calc(db = db,
                            x = rf.i,
                            y = "Creditability")
      woe.tbl[[i]] <- cbind.data.frame(rf = rf.i,
                                       woe.tbl.i[[1]][, c("bin", "woe")])
      }
woe.tbl <- bind_rows(woe.tbl)
#print observed woe table
woe.tbl
```

```
##                rf         bin        woe
## 1 Account_Balance          01 -0.8180987
## 2 Account_Balance          02 -0.4013918
## 3 Account_Balance          03  0.4054651
## 4 Account_Balance          04  1.1762632
## 5        Maturity 01 (-Inf,8)  1.3121864
## 6        Maturity   02 [8,16)  0.3466246
## 7        Maturity  03 [16,36) -0.1086883
## 8        Maturity  04 [36,45) -0.5245245
## 9        Maturity 05 [45,Inf) -1.1349799
```

```
#manual woe adjustment
woe.tbl$woe[3] <- 0
#print adjusted woe table
woe.tbl
```

```
##                rf         bin        woe
## 1 Account_Balance          01 -0.8180987
## 2 Account_Balance          02 -0.4013918
## 3 Account_Balance          03  0.0000000
## 4 Account_Balance          04  1.1762632
## 5        Maturity 01 (-Inf,8)  1.3121864
## 6        Maturity   02 [8,16)  0.3466246
## 7        Maturity  03 [16,36) -0.1086883
## 8        Maturity  04 [36,45) -0.5245245
## 9        Maturity 05 [45,Inf) -1.1349799
```

With the above inputs prepared, practitioners can run the same model shift simulations and compare the

results. In this run, we will again compare the matrix multiplication and the weighted quasi-binomial methods.

1. Matrix Multiplication with Manual WoE Encoding:

```
res.mm <- ms.pd(db = db,
                target = "Creditability",
                rf = c("Account_Balance", "Maturity"),
                data.shift = data.shift,
                encoding = "WoE",
                method =  "mm",
                woe.tbl = woe.tbl)
```

2. Weighted Quasi-Binomial Regression with Manual WoE Encoding:

```
res.wf <- ms.pd(db = db,
                target = "Creditability",
                rf = c("Account_Balance", "Maturity"),
                data.shift = data.shift,
                encoding = "WoE",
                method =  "wfr",
                woe.tbl = woe.tbl)
```

The following data frame compares the results from the above runs.

```
data.frame(mm.woe = res.mm$ms,
           wf.woe = res.wf$ms)
```

```
##                            mm.woe      wf.woe
## (Intercept)         -0.0007908939 0.001732708
## Account_Balance.woe  0.0154189541 0.015969937
## Maturity.woe         0.0371189774 0.034682209
```

To conclude this section, let's run another simulation for the model with multiple risk factors. For this purpose, we will import a new dataset and run the simulation. To align with the design most commonly used in practice, we will opt for `"WoE"` encoding, while for the data shift, we will use the same scenario as in the previous simulations.

```
#import dataset
fp <- "https://raw.githubusercontent.com/andrija-djurovic/adsfcr/main/mrm/ms_db_1.csv"
db <- read.csv(file = fp,
               colClasses = "character",
               header = TRUE)
#convert target to numeric
db$Creditability <- as.numeric(db$Creditability)
#check dataset structure
str(db)
```

```
## 'data.frame':    1000 obs. of  7 variables:
##  $ Creditability                   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Account_Balance                 : chr  "01" "01" "02" "01" ...
##  $ Duration_of_Credit__month       : chr  "03 [16,45)" "02 [8,16)" "02 [8,16)" "02 [8,16)" ...
##  $ Payment_Status_of_Previous_Credit: chr  "04" "04" "02" "04" ...
##  $ Purpose                         : chr  "02" "00" "09" "00" ...
##  $ Value_Savings_Stocks            : chr  "01" "01" "02" "01" ...
##  $ Length_of_current_employment    : chr  "02" "03" "04" "03" ...
```

```
#print data shift input
data.shift
```

```
##   Account_Balance     n
## 1              01 383.6
## 2              02 269.0
## 3              03  63.0
## 4              04 284.4
```

1. Matrix Multiplication with WoE Encoding:

```
res.mm <- ms.pd(db = db,
                target = "Creditability",
                rf = names(db)[-1],
                data.shift = data.shift,
                encoding = "WoE",
                method =  "mm",
                woe.tbl = NULL)
```

2. Weighted Quasi-Binomial Regression with WoE Encoding:

```
res.wf <- ms.pd(db = db,
                target = "Creditability",
                rf = names(db)[-1],
                data.shift = data.shift,
                encoding = "WoE",
                method =  "wfr",
                woe.tbl = NULL)
```

The following data frame compares the results from the above runs.

```
data.frame(mm.woe = res.mm$ms,
           wf.woe = res.wf$ms)
```

```
##                                          mm.woe       wf.woe
## (Intercept)                         -0.006661372 -0.004677474
## Account_Balance.woe                  0.012746237  0.008321911
## Duration_of_Credit__month.woe        0.022230188  0.017521925
## Payment_Status_of_Previous_Credit.woe  0.029643169  0.027173187
## Purpose.woe                         -0.075207284 -0.068414196
## Value_Savings_Stocks.woe             0.046837955  0.038668358
## Length_of_current_employment.woe    -0.171941549 -0.158032535
```

6 Future Development

The `ModelShift` package is currently in the development phase. Although the current version supports the most commonly used methods, future versions can still be improved.

One feature currently not supported by the package is data shift based on combinations of risk drivers that do not appear in the supplied modeling dataset (`db`). This means if such a combination exists in the dataset provided for the `data.shift` argument, it will be ignored when quantifying the model parameter shift.

Practitioners should also consider performance optimization when running a very high number of simulations. For example, the current version of the package allows supplying an already estimated initial model via the `lr.i` argument, which is recommended to avoid re-estimating the initial model in every simulation. Certain conditions within the existing function implementation can be removed or adjusted depending on the design to optimize the simulation process.

Another improvement being considered for the upcoming version involves introducing statistical hypothesis testing for the shift in quantified parameters.

Additionally, similar methods can be extended to Loss Given Default (LGD) and Exposure at Default (EaD) models, which often use linear regression to estimate the risk differentiation function.

Finally, a `Python` version of this `R` package is in the pipeline and will be developed in the coming months.

Beyond these improvements, practitioners should also be aware of specific data requirements when using the existing model. For instance, the data frame supplied to the `data.shift` argument must contain the same modalities as those in the modeling dataset (`db`). Otherwise, the function will return an error.

Another aspect worth mentioning relates explicitly to the `mm` (matrix multiplication) method. To enable the matrix multiplication process, zero model points (the diagonal of matrices Y^+ and Y^-) were replaced with `1e-10`. If practitioners prefer a different approach or wish to assess the bias introduced by this imputation step, they are encouraged to modify the source code of this method directly.