Business-Guided Regression Designs

Statistical Approach to Third Party Ratings Treatment: Constrained Threshold Logistic Regression

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Third Party Support in Credit Risk

- The ownership structure of a corporate entity significantly influences its creditworthiness.
- Third party support can strengthen a company's credit profile if the supporting party is financially stable and willing to assist during times of stress.
- Conversely, if the third party has a weaker credit profile and relies on the subsidiary's cash flows to meet its obligations, its risk profile may deteriorate.
- Third party support can take various forms and be shaped by multiple factors.
- When assessing its impact on rating assignments, practitioners typically evaluate, among other factors, the timeliness and adequacy of this support in preventing default.
- Regulatory frameworks recognize the role of third party support and guide the incorporation of third party ratings into credit risk assessments.

Statistical Approach to Third Party Ratings Treatment

- Unlike other standardized statistical approaches for model development, practitioners must consider additional inputs and modeling constraints when treating third party rating modeling.
- Given the variety of supporting structures between the third party and subsidiary and data availability, developing a modeling framework that suits all situations is challenging. Therefore, practitioners often opt for tailor-made modeling approaches when tackling this exercise.
- The following slide outlines one of the approaches proposed by Andrija Djurovic, which can serve as a basis for modeling third party support.
- Like other approaches, this relies on specific assumptions, such as the availability and comparability of scores between the third party and subsidiary. It also assumes that the new score always lies between the third party and subsidiary scores.
- The following slide provides a brief description of the presented method, which can be considered another tailor-made regression design, along with the `R` and `Python` code for its implementation.

Statistical Approach to Third Party Ratings Treatment cont.

The following points outline the steps for a statistical approach to third party rating modeling under the assumptions mentioned on the previous slide:

1. With both subsidiary (ss_{score}) and the third party (gr_{score}) scores available, estimate a logistic regression model of the form:

$$y \sim \alpha + \beta \cdot \text{Score}$$

   where y is the default indicator, and α and β are the regression estimates.
2. Using the results from step 1, compute the log odds separately for the subsidiary ($ss.lo$) and the third party ($gr.lo$).
3. Following the idea of threshold models, create two variables that reflect cases where the subsidiary score is lower than the third party score - n - and cases where the subsidiary score is greater than or equal to the third party score - p. The variables n and p are defined as the absolute difference between the log odds obtained in step 1.
4. To estimate potential improvements in the subsidiary score given the third party score, fit the following constrained logistic regression:

$$y \sim \beta_1 \cdot n + \beta_2 \cdot p$$

   while setting the offset equal to the subsidiary log-odds ($ss.lo$) and imposing the constraints $-1 < \beta_1 < 0$ and $0 < \beta_2 < 1$. By applying these constraints along with the offsets, the predicted probability always falls between those of the subsidiary and the third party.

Practitioners should remember that, depending on the relationship between the initial scores and the default indicator (y), the constraints in step 4 may need to be reversed.

The following slides present the results of a simulation study conducted on the dataset available here.

R Code

```r
#data import
url <- "https://raw.githubusercontent.com/andrija-djurovic/adsfcr/main/model_dev_and_vld/db_tps.csv"
db <- read.csv(file = url,
               header = TRUE)

#step 1: individual model estimation
lr.ss <- glm(formula = default ~ ss_score,
             family = "binomial",
             data = db)
round(x = summary(lr.ss)$coefficients, digits = 4)
```

```
##             Estimate Std. Error  z value Pr(>|z|)
## (Intercept)   2.0341     0.3496   5.8179        0
## ss_score     -0.0059     0.0005 -12.9350        0
```

```r
lr.gr <- glm(formula = default ~ gr_score,
             family = "binomial",
             data = db)
round(x = summary(lr.gr)$coefficients, digits = 4)
```

```
##             Estimate Std. Error  z value Pr(>|z|)
## (Intercept)   2.6401     0.3780   6.9842        0
## gr_score     -0.0066     0.0005 -13.3741        0
```

```r
#step 2: log-odds
db$ss.lo <- predict(object = lr.ss)
db$gr.lo <- predict(object = lr.gr)
#step 3: absolute difference
dist <- abs(db$ss.lo - db$gr.lo)
```

R Code cont.

```r
#step 3: treshold variables
db$n <- (db$ss_score < db$gr_score)  * dist
db$p <- (db$ss_score >= db$gr_score) * dist

#step 4: the log-likelihood of constrained logistic regression
ll <- function(beta, X, y, offset) {
      lp <- X %*% beta + offset
      fitted <- exp(lp) / (1 + exp(lp))
      opt.f <- -sum(y * log(fitted) + (1 - y) * log(1 - fitted))
return(opt.f)
}

#step 4: optimization process
X <- as.matrix(data.frame(db$n, db$p))
y <- as.matrix(db$default)
offset <- db$ss.lo
opt.res <- optim(par = c(0, 0),
                 fn = ll,
                 X = X,
                 y = y,
                 offset = offset,
                 lower = c(-1, 0),
                 upper = c(0, 1),
                 method = "L-BFGS-B")

#step 4:optimized coefficients
beta.est <- opt.res$par
names(beta.est) <- c("n", "p")
beta.est
```

```
##          n          p
## -0.6962447  0.7640713
```

Python Code

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
import statsmodels.formula.api as smf
from scipy.optimize import minimize

#data import
url = "https://raw.githubusercontent.com/andrija-djurovic/adsfcr/main/model_dev_and_vld/db_tps.csv"
db = pd.read_csv(filepath_or_buffer = url)

#step 1: individual model estimation
lr_ss = smf.glm(formula = "default ~ ss_score",
                family = sm.families.Binomial(),
                data = db).fit()
print(round(lr_ss.summary2().tables[1], 4))
```

```
##             Coef.  Std.Err.        z  P>|z|  [0.025  0.975]
## Intercept  2.0341    0.3496   5.8179    0.0  1.3488  2.7193
## ss_score  -0.0059    0.0005 -12.9350    0.0 -0.0068 -0.0050
```

```
lr_gr = smf.glm(formula = "default ~ gr_score",
                family = sm.families.Binomial(),
                data = db).fit()
print(round(lr_gr.summary2().tables[1], 4))
```

```
##             Coef.  Std.Err.        z  P>|z|  [0.025  0.975]
## Intercept  2.6401    0.3780   6.9842    0.0  1.8992  3.3810
## gr_score  -0.0066    0.0005 -13.3741    0.0 -0.0076 -0.0056
```

```
#step 2: log-odds (linear predictor)
db["ss_lo"] = lr_ss.predict(which = "linear")
db["gr_lo"] = lr_gr.predict(which = "linear")
```

Python Code cont.

```python
#step 3: absolute difference
dist = np.abs(db["ss_lo"] - db["gr_lo"])

#step 3: threshold variables
db["n"] = (db["ss_score"] < db["gr_score"]) * dist
db["p"] = (db["ss_score"] >= db["gr_score"]) * dist

#step 4: the log-likelihood of constrained logistic regression
def ll(beta, X, y, offset):
    lp = np.dot(X, beta) + offset
    fitted = np.exp(lp) / (1 + np.exp(lp))
    log_likelihood = -np.sum(y * np.log(fitted) + (1 - y) * np.log(1 - fitted))
    return log_likelihood

#step 4: optimization process
X = db[["n", "p"]].values
y = db["default"].values
offset = db["ss_lo"].values
opt_res = minimize(fun = ll,
                   x0 = np.array([0, 0]),
                   args = (X, y, offset),
                   method = "L-BFGS-B",
                   bounds = [(-1, 0), (0, 1)])

#step 4: optimized coefficients
beta_est = opt_res.x
pd.Series(beta_est, index = ["n", "p"])
```

```
## n   -0.696244
## p    0.764071
## dtype: float64
```