Statistical Binning and Model Validation

How the Choice of Binning Algorithm Influences Model Validation

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Statistical Binning in Credit Risk

- Binning is not a mandatory step in model development but offers several advantages over using continuous risk factors.
- Acknowledging its benefits, practitioners often employ binning while developing credit risk models.
- The most commonly used principles of a good binning process are:
  - each bin should contain at least 5% of the observations;
  - each bin should contain at least one bad case;
  - adjacent bins should have different riskiness levels;
  - the risk levels of the bins should follow either a monotonic or U-shape trend;
  - the number of bins should not be greater than ten.
- Refer to this document for more details on the statistical binning of numeric risk factors.
- In practice, questions often arise about whether all the principles must be followed.
- It is also crucial to understand how binning connects to other modeling steps and validation processes.

Practical Challenges with the Binning Process

- Binning can be applied to both numeric and categorical risk factors, with numeric factors often being the focus of automated statistical binning procedures.
- The binning process for numeric risk factors typically follows a monotonic or U-shaped trend.
- In practice, there are various ways to perform binning for risk factors, leading to different levels of granularity in the final number of bins.
- Although there are no formal guidelines on which binning algorithms to use, their selection should consider a combination of statistical and business objectives.
- One of the biggest challenges in the model validation process is addressing binning procedures that produce a "large" number of bins.
- Such risk drivers are often deemed unstable by validators, but is that inherently a problem?
- The following slides present a simplified simulation design demonstrating how different binning algorithms can impact the final rating scale.

Simulation Design and Expected Results

Simulation Design

The following steps outline a simplified simulation demonstrating how different binning algorithms affect the number of rating grades derived from various models:

- Assume a development sample of 1,000 observations, with the target variable being a default indicator (`default`) and three numeric risk factors (`maturity`, `age`, `amount`).
- We categorized the numeric risk factors using two different monotonic binning algorithms. For demonstration purposes, the following slides utilize two functions implemented identically in `R` and `Python`: `iso.bin` and `sts.bin` from the `monobin` package in `R`, and `iso_bin` and `sts_bin` from the `monobinpy` package in `Python`.
- Due to their differing underlying algorithms, these two functions are expected to produce different numbers of bins for some or all risk factors undergoing the binning process.
- After binning and creating new discretized risk factors (`maturity.iso`, `age.iso`, `amount.iso`, `maturity.sts`, `age.sts`, `amount.sts`), we ran two logistic regressions using the discretized risk factors from each binning procedure.
- Finally, we generate within-sample predictions from both regressions and apply `sts.bin/sts_bin` to create the final rating scale.

Expected results

As the results on the following slide demonstrate, the two models produce different numbers of final rating grades.

Given these differences in the final output, can we definitively determine whether one binning algorithm is superior?

R Code

```r
library(monobin)

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

#numeric risk factors
rf <- names(db)[-1]

#----------------------------iso.bin algorithm----------------------------#
iso.b <- sapply(X = rf, FUN = function(x) {
                              iso.bin(x = db[, x], y = db$default)[[2]]})
#prepare data frame
iso.b <- data.frame(iso.b)
names(iso.b) <- c("maturity.iso", "age.iso", "amount.iso")
#check unique number of bins per risk factor
sapply(X = iso.b, FUN = function(x) length(unique(x)))
```

```
## maturity.iso      age.iso   amount.iso
##            7            4            3
```

```r
#add discretized risk factors
db <- cbind.data.frame(db, iso.b)

#----------------------------sts.bin algorithm----------------------------#
sts.b <- sapply(X = rf, FUN = function(x) {
                              sts.bin(x = db[, x], y = db$default)[[2]]})
#prepare data frame
sts.b <- data.frame(sts.b)
names(sts.b) <- c("maturity.sts", "age.sts", "amount.sts")
#check unique number of bins per risk factor
sapply(X = sts.b, FUN = function(x) length(unique(x)))
```

```
## maturity.sts      age.sts   amount.sts
##            5            3            3
```

```r
#add discretized risk factors
db <- cbind.data.frame(db, sts.b)
```

R Code cont.

```
#------------------------------model comparison-----------------------------#
#run glm based on iso.bin risk factors
lr.1 <- glm(formula = default ~ `maturity.iso` + `age.iso` + `amount.iso`,
            family = binomial,
            data = db)
#run glm based on sts.bin risk factors
lr.2 <- glm(formula = default ~ `maturity.sts` + `age.sts` + `amount.sts`,
            family = binomial,
            data = db)

#create rating scale for lr.1 based on sts.bin algorithm
sts.bin(x = predict(object = lr.1), y = db$default)[[1]][, 1:4]
```

```
##                     bin  no y.sum      y.avg
## 1    01 (-Inf,-1.9637)  73     4 0.05479452
## 2 02 [-1.9637,-1.3642) 181    30 0.16574586
## 3 03 [-1.3642,-0.9815) 208    51 0.24519231
## 4 04 [-0.9815,-0.3044) 335   113 0.33731343
## 5  05 [-0.3044,0.0315) 128    56 0.43750000
## 6      06 [0.0315,Inf)  75    46 0.61333333
```

```
#create rating scale for lr.2 based on sts.bin algorithm
sts.bin(x = predict(object = lr.2), y = db$default)[[1]][, 1:4]
```

```
##                     bin  no y.sum      y.avg
## 1    01 (-Inf,-1.9011)  73     4 0.05479452
## 2 02 [-1.9011,-1.5505) 152    23 0.15131579
## 3 03 [-1.5505,-1.0054) 233    58 0.24892704
## 4 04 [-1.0054,-0.2518) 362   123 0.33977901
## 5     05 [-0.2518,Inf) 180    92 0.51111111
```

Python Code

```python
import pandas as pd
import numpy as np
from monobinpy import iso_bin, sts_bin
import statsmodels.api as sm
import statsmodels.formula.api as smf

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

#numeric risk factors
rf = db.columns[1:]

#----------------------------iso_bin algorithm----------------------------#
iso_b = {x: iso_bin(x = db[x], y = db["default"])[1] for x in rf}
#prepare data frame
iso_b = pd.DataFrame(iso_b)
iso_b.columns = ["maturity.iso", "age.iso", "amount.iso"]

#check unique number of bins per risk factor
{col: iso_b[col].nunique() for col in iso_b.columns}
```

```
## {'maturity.iso': 7, 'age.iso': 4, 'amount.iso': 3}
```

```python
#add discretized risk factors to the original data
db = pd.concat([db, iso_b], axis = 1)

#----------------------------sts_bin algorithm----------------------------#
sts_b = {x: sts_bin(x = db[x], y = db["default"])[1] for x in rf}
#prepare data frame
sts_b = pd.DataFrame(sts_b)
sts_b.columns = ["maturity.sts", "age.sts", "amount.sts"]

#check unique number of bins per risk factor
{col: sts_b[col].nunique() for col in sts_b.columns}
```

```
## {'maturity.sts': 5, 'age.sts': 3, 'amount.sts': 3}
```

```python
#add discretized risk factors to the original data
db = pd.concat([db, sts_b], axis = 1)
```

Python Code cont.

```python
# -----------------------------model comparison----------------------------#
#run glm based on iso_bin risk factors
formula_iso = "default ~ Q('maturity.iso') + Q('age.iso') + Q('amount.iso')"
lr_1 = smf.glm(formula = formula_iso,
               data = db,
               family = sm.families.Binomial()).fit()

#run glm based on sts_bin risk factors
formula_sts = "default ~ Q('maturity.sts') + Q('age.sts') + Q('amount.sts')"
lr_2 = smf.glm(formula = formula_sts,
               data = db,
               family = sm.families.Binomial()).fit()

#create rating scale for lr_1 based on sts_bin algorithm
pred_lr_1 = pd.Series(lr_1.predict(linear = True))
sts_bin(x = pred_lr_1, y = db["default"])[0].iloc[:, :4]
```

```
##                      bin   no  y_sum     y_avg
## 0     01 (-inf, -1.9637)   73      4  0.054795
## 1  02 [-1.9637, -1.3642)  181     30  0.165746
## 2  03 [-1.3642, -0.9815)  208     51  0.245192
## 3  04 [-0.9815, -0.3044)  335    113  0.337313
## 4   05 [-0.3044, 0.0315)  128     56  0.437500
## 5       06 [0.0315, inf)   75     46  0.613333
```

```python
#create rating scale for lr_2 based on sts_bin algorithm
pred_lr_2 = pd.Series(lr_2.predict(linear = True))
sts_bin(x = pred_lr_2, y = db["default"])[0].iloc[:, :4]
```

```
##                      bin   no  y_sum     y_avg
## 0     01 (-inf, -1.9011)   73      4  0.054795
## 1  02 [-1.9011, -1.5505)  152     23  0.151316
## 2  03 [-1.5505, -1.0054)  233     58  0.248927
## 3  04 [-1.0054, -0.2518)  362    123  0.339779
## 4      05 [-0.2518, inf)  180     92  0.511111
```

Discussion Points

- How should model validators (internal, external, or regulators) analyze models containing discretized risk factors with an excessive number of bins?
- Should validators automatically penalize the potential instability of discretized risk factors with an excessive number of bins?
- What steps should modelers take during the development process to anticipate potential issues during validation (both initial and periodic)?
- What are the differences between using numeric risk factors and those discretized with an excessive number of bins?
- How should validators address this issue when dealing with machine learning models used in credit risk modeling?