Binnr Quick Start Guide

## Installation

The freshest way to install binnr is from Gitlab. This requires two support packages that should be installed anyhow: installr & devtools.

if(!require(installr)) install.packages("installr")

if(!require(devtools)) install.packages("devtools")

installr::install.Rtools() type=”binary”)

url <- "https://gitlab.ins.risk.regn.net/minneapolis-r-packages/"

devtools::install\_git(paste0(url, "binnr.git"), build\_vignettes=TRUE)

devtools::install\_git(paste0(url, "binnrtools.git"), build\_vignettes=TRUE)

devtools::install\_git(paste0(url, "mkivtools.git"), build\_vignettes=TRUE)

The first step in any binnr development is to prepare the variables for modeling. The bin function processes a data.frame for downstream modeling and manipulation.

The resulting object is a Scorecard object. It stores everything about a model and provides methods for manipulating the Scorecard.

## Classing Variables

library(binnr)

library(mkivtools)

library(binnrtools)

register\_mkiv("Z:/Resources/\_MKIV/consumer-mkiv/mk\_iv\_5\_2\_3.sas")

d <- read.csv("mkiv\_perf.csv", header=TRUE)

mod <- bin(d, d$depvar, exceptions=-1, mono=2, min.res=25, min.cnt=100)

Binning : ==========|

Warning messages:

1: dropping variables with all NA values: nf\_inq\_adls\_per\_email, nf\_email\_name\_addr\_ver

2: Variable, account, has more than 20 levels -- Skipping

3: Variable, nf\_fp\_addrchangeecontraj, has more than 20 levels -- Skipping

## Correlated Predictors

The Scorecard variables can be clustered into groups with related correlation coefficients.

We can subsequently view all of the variable cluster groups or prune the groups retaining a representative variable from each one. The retained variables have the highest information value within each cluster.

cc <- mod$cluster(bag.fraction = 0.20)

clusters <- mod$get\_clusters(cc, corr = 0.80)

> head(clusters)

variable sort\_value Cluster

1 rv\_L79\_adls\_per\_apt\_addr\_c6 0.05604534 1

2 rv\_L79\_adls\_per\_apt\_addr 0.05559702 1

3 nf\_inq\_ssns\_per\_sfd\_addr 0.05299009 1

4 nf\_inq\_lnames\_per\_apt\_addr 0.05073883 1

5 nf\_inq\_adls\_per\_apt\_addr 0.05019285 1

6 nf\_inq\_per\_apt\_addr 0.04940860 1

drop\_corr <- mod$prune\_clusters(cc, corr=0.80, n=1)

mod$drop(drop\_corr)

## Fit Initial Model

mod$fit(“model 1”, “initial model with all variables”)

Dropping highly correlated variables is a good first step before fitting an initial model.

A good second step is repeatedly fitting the model on bootstrap samples to winnow the candidate set down even further.

The boot strap sample results reveal variables that have high-variance coefficients or enter the model sporadically. For example, some variables only have non-zero coefficients in 1 out of 20 model runs and are clearly not reliable.

> mod

2 models

|-- scratch | 00.0 ks |

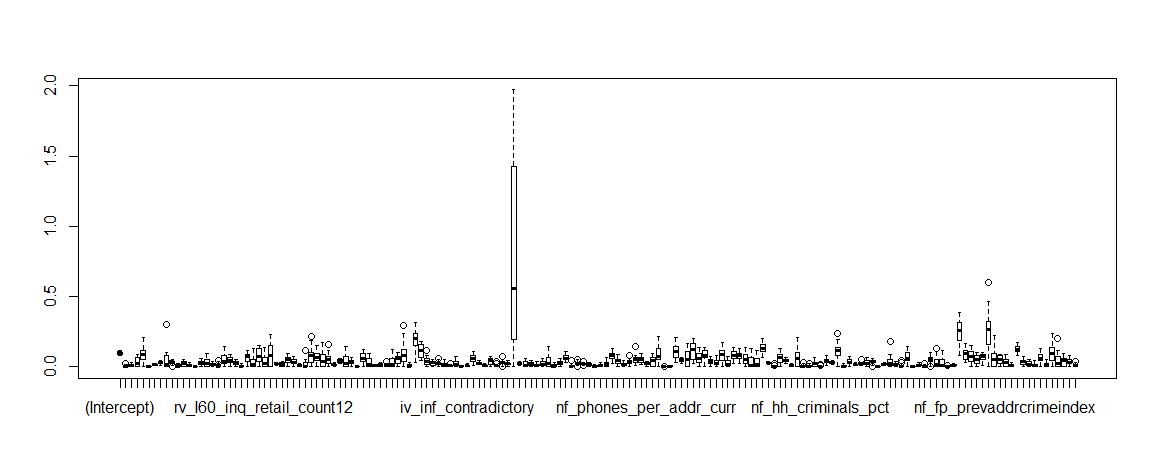
|-- model 1 | 47.3 ks | initial model with all variables

mod$drop(mod$get\_inmodel(invert = TRUE))

## Bootstrap Model Fits

pvals <- mod$pseudo\_pvalues(20, bag.fraction = 1, replace = TRUE)

boxplot(t(pvals$coefs)



v <- apply(pvals$coefs, 1, var) # some coefs have high variance

We can pass the names of these unreliable variables to the `drop` method and refit the model. The out-of-fold KS improved considerably. We now have a solid set of candidate variables to investigate through the `adjust` method shown below.

high\_var <- names(which(v > quantile(v, 0.95)))

high\_pval <- names(which(pvals$pvalues > 0.10)) # some enter the model inconsistently

mod$drop(c(high\_var, high\_pval))

mod$fit("model 2", "after dropping high variance/pvalue vars")

> mod

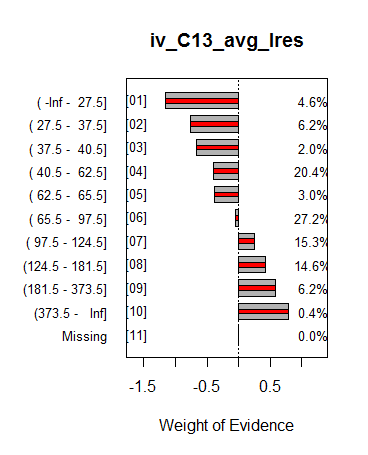
2 models

|-- scratch | 00.0 ks |

|-- model 1 | 47.3 ks | initial model with all variables

|-- \* model 2 | 49.2 ks | after dropping high variance/pvalue vars

## Adjust the Model Variables

mod$adjust()

iv\_C13\_avg\_lres

N #1 #0 %N %1 %0 P(1) WoE IV Pred

[01] ( -Inf - 27.5] 460 10 450 0.046 0.015 0.048 0.022 -1.155 0.038 -1.155

[02] ( 27.5 - 37.5] 623 20 603 0.062 0.030 0.065 0.032 -0.755 0.026 -0.755

[03] ( 37.5 - 40.5] 201 7 194 0.020 0.011 0.021 0.035 -0.671 0.007 -0.671

[04] ( 40.5 - 62.5] 2045 92 1953 0.204 0.140 0.209 0.045 -0.404 0.028 -0.404

[05] ( 62.5 - 65.5] 303 14 289 0.030 0.021 0.031 0.046 -0.376 0.004 -0.376

[06] ( 65.5 - 97.5] 2716 172 2544 0.272 0.261 0.272 0.063 -0.043 0.000 -0.043

[07] ( 97.5 - 124.5] 1534 127 1407 0.153 0.193 0.151 0.083 0.246 0.010 0.246

[08] (124.5 - 181.5] 1456 142 1314 0.146 0.215 0.141 0.098 0.426 0.032 0.426

[09] (181.5 - 373.5] 625 70 555 0.062 0.106 0.059 0.112 0.581 0.027 0.581

[10] (373.5 - Inf] 37 5 32 0.004 0.008 0.003 0.135 0.795 0.003 0.795

[11] Missing 0 0 0 0.000 0.000 0.000 0.000 0.000 0.000 0.000

Total 10000 659 9341 1.000 1.000 1.000 0.000 0.000 0.176 0.000

[In Model: TRUE | Dropped: FALSE]

Enter command (Q to quit; h for help):

?

## Bring up Variable Definition

If a MKIV is registered through the `mkivtools` library, typing a “?” during the interactive adjust session will pull up the model code the Rstudio Viewer pane.

\*# iv\_C13\_avg\_lres;

\*@group: Length of Residence;

\*@description: Average Length of residence;

if not truedid

then iv\_C13\_avg\_lres = .;

else iv\_C13\_avg\_lres = min(avg\_lres, 999);

## Apply Bin Operations

The adjust method starts an interactive session where the modeler can enter commands to manipulate binned variables. Bins can be expanded, collapsed, and neutralized much like in Xeno.

Variables should be inspected for palatability and regulatory conformance. Once they are adjusted to satisfaction, the model is nearly finished.

|  |  |
| --- | --- |
| Command | Definition |
| (Q)uit | Quit adjust function |
| (n)ext | Move to next variable |
| (p)revious | Move to previous variable |
| (g)oto | Goto variable; prompted to enter variable name |
| (m)ono | Change monotonicity when prompted |
| (e)xceptions | Change variable exceptions when prompted |
| (s)et equal | Set one WoE level equal to another when prompted |
| (u)ndo | Undo the last manipulation command |
| (r)eset | Reset the bin to its initial state |
| (d)rop/undrop | Flag the variable as dropped or un-dropped |
| != <#> | Neutralize requested variable levels (WoE -> 0) |
| + <#> | Expand requested level (one at a time) |
| - <#> | Collapse requested levels |

## Finalize Model

A final round of bootstrap model fitting serves identify truly predictive from spurious variables.

pvals <- mod$pseudo\_pvalues(50, bag.fraction = 1, replace = TRUE)

high\_pval <- names(which(pvals$pvalues > 0.10))

mod$drop(high\_pval)

mod$fit("model 3", "Final Model", nfolds = 20)

> mod

A combination of model adjustments and further winnowing increase the KS by another point.

4 models

|-- scratch | 00.0 ks |

|-- model 1 | 47.3 ks | initial model with all variables

|-- model 2 | 49.2 ks | after dropping high variance/pvalue vars

|-- \* model 3 | 50.5 ks | Final Model

## Analyze Model Variables

There are several functions for analyzing the final model. Univariate statistics can be reported using the `summary` method.

Different models can be compared using the `compare` method.

If a MKIV is registered, variable groups missing from the model can be easily identified and inspected.

s <- mod$summary(inmodel.only=TRUE) ## data.frame summarizing model variables

> mod$compare(“model 2”, “model 3”)

mkiv\_summary <- summarize\_model\_vars(mod)

> View(mkiv\_summary$missing\_groups)

> View(mkiv\_summary$model\_vars)

## Generate SAS Code

code <- mod$gen\_code\_sas(pfx = “mod1”, method=”neutral”)

cat(code, file=”my\_binnr\_model.sas”, sep=”\n”)

/\*\*\* nf\_fp\_curraddrmurderindex \*\*\*/

if missing(nf\_fp\_curraddrmurderindex)

then mod1\_V01\_w = 0;

If I MKIV is registered, binnr will print the mkiv variable code in the generated SAS code.

else if nf\_fp\_curraddrmurderindex <= 1.5

then mod1\_V01\_w = -0.902187998182727;

else if nf\_fp\_curraddrmurderindex <= 46.5

then mod1\_V01\_w = -0.0974965223769837;

else if nf\_fp\_curraddrmurderindex <= 93.5

then mod1\_V01\_w = -0.0879939778663614;

else if nf\_fp\_curraddrmurderindex <= 152.5

then mod1\_V01\_w = 0.0817707163869025;

else mod1\_V01\_w = 0.142064000910119;

## Exporting Bivariates

lm <- export\_classing(mod, sheet = "bivariates")

lm$open\_workbook()

Pretty bivariates can all be sent directly to Excel if the binnrtools package is loaded.

These functions can take a while to run.