Binnr cheat sheet

Installation

```
install.packages(”path/to/binnr.zip”, repos=NULL, type=”binary”)
> require(binnr)
> data(titanic) # data set used for all examples
```

Binning Data

```
> mod <- bin(data=titanic, y=titanic$Survived)
```

Optional Bin Function Arguments

Argument	Definition
data	data.frame of independent predictors
y	Performance variable (binary only for now)
w	Numeric vector of weights
min.iv	Minimum information gain for a split
min.cnt	Minimum number of records per bin
min.res	Minimum number of responses per bin
max.bin	Maximum number of bins
mono	Monotonicity: • -1 Decreasing • 0 No monotonicity • 1 Increasing • 2 Either increasing or decreasing
exceptions	Values to withhold from discretization

Handling Multi-collinearity

```
> cc <- mod$cluster()
> mod$get_clusters(cc, corr=0.60) # returns data.frame of vars
  variable sort_value Cluster
1     Fare 0.74722120       1
2   Pclass 0.50094974       1
3      Sex 1.34168141       2
> to_drop <- mod$prune_clusters(cc, corr=0.60, n=1)
> to_drop
"Pclass"
> mod$drop(to_drop)
```

Inspect/Alter Variables in the Classing

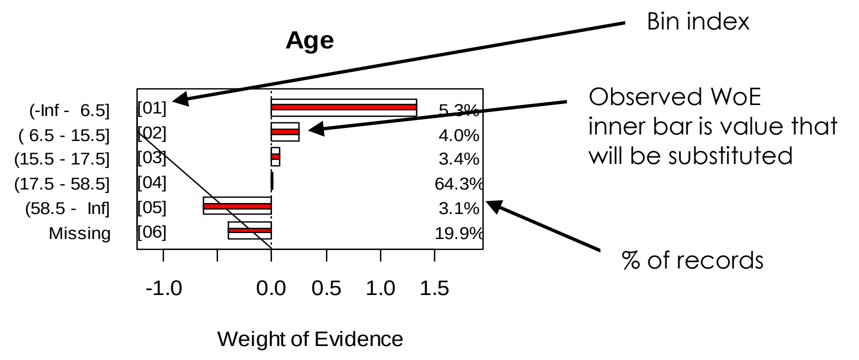
```
> other <- bin(new_vars, y=titanic$Survived)
> mod$combine(other)
> mod$get_dropped(invert = FALSE) # optionally invert selection
> mod$get_inmodel(invert = FALSE) # optionally invert selection

> mod$drop(c("Pclass", "Fare"))
> mod$get_dropped()
[1] "Survived" "Pclass"   "Fare"
```

Viewing Data

This is typically handled through the `adjust` function.

```
> mod$drop(to_drop)
> mod$variables$Age$show()
```



Fitting Models

```
> mod$fit(“model 1”, “Initial model with all variables”)
> mod
2 models
 |--    scratch     | 00.0 ks |
 |-- *  model 1     | 58.2 ks | Initial model with all variables
```

Reviewing Models

```
> mod$sort() # sort by inmodel, not dropped, then IV
> mod$summary(inmodel.only=TRUE) # print summary statistics
> mod$compare(“model 1”, “model 2”) # compare coefs & contrib
```

Switching Models

All binnr commands are relative to the currently selected model. This model has an asterisk next to it in the model printout.

```
> mod$select(“scratch”)
> mod
2 models
 |-- *  scratch     | 00.0 ks |
 |--    model 1     | 58.2 ks | Initial model with all variables
```

Adjusting Bins

```
mod$adjust() ## this is the workhorse function of `binnr`
mod$adjust(“Pclass”) ## Start at variable Pclass

## Can also do outside of the adjust function (not recommended)
mod$variables$Pclass$collapse(1:2)
mod$variables$Fare$mono(2)
mod$variables$Age$exceptions(c(-1,-2))
```

Bin Manipulation Commands

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 level(s): - Adjacent for Continuous bins (ex: - 1:2) - Can be separate for Discrete bins (ex: - c(1,3))

All of the bin manipulation functions modify the Scorecard object in place.

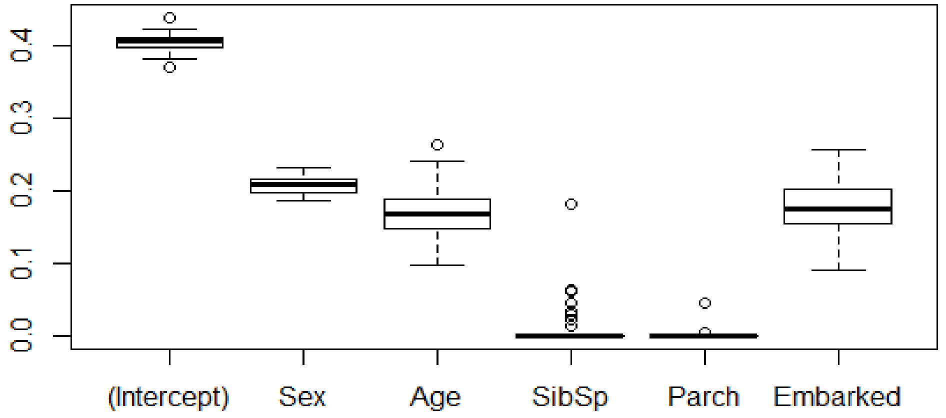
Pseudo P-Values

When the model is mostly finished run several bootstrap fits to determine which variables are entering by chance and which enter repeatedly.

```
> pvals <- mod$pseudo_pvalues(times = 50, bag.fraction = 1,
                              replace = TRUE)
```

Argument	Definition
times	Number of repeated samples to draw
bag.fraction	Percent of records to sample
replace	Whether to sample with replacement

```
> boxplot(t(pvals$coefs))
```



```
> pvals$pvalues
(Intercept)        Sex        Age      SibSp      Parch   Embarked
       0.00       0.00       0.00       0.84       0.96       0.00
> mod$drop(names(which(pvals$pvalues > 0.10)))
```

The pseudo p-values represent the percentage of model runs that the coefficient was zero. Dropping all variables that exceed a pseudo p-value threshold and refitting removes spurious variables and results in a parsimonious model.

```
> mod$fit(“final model”)
> mod
3 models
 |--    scratch                | 00.0 ks |
 |--    model 1                | 58.3 ks | initial model
 |-- *  final model            | 57.1 ks |
```

The out-of-fold KS drops a little, but the tradeoff is likely worth it for a scorecard with fewer variables.

Making Predictions

Binnr can return score predictions or a matrix of weight-of-evidence substitutions. Can also predict on new data sets.

```
> p <- mod$predict()
> p[1:4]
[1] -2.3726886  2.7146116  0.2156492  2.0621833
> p_val <-mod$predict(newdata=validation_data)
> woe <- mod$predict(type="woe")
> woe[1:2,1:4]
         Pclass        Sex        Age     SibSp
[1,] -0.6664827 -0.9838327 0.01517886 0.3388098
[2,]  1.0039160  1.5298770 0.01517886 0.3388098
```

Generating SAS Code

Binnr provides functions for generating SAS model code.

```
> code <- mod$gen_code_sas(pfx="mod1")
> cat(code, sep="\n", file=”my_sas_code.sas”)
```

Argument	Definition
pfx	Prefix to append to model variable names
method	Adverse Action code calculation method” 1. Min – Points from min bin value 2. Max – Points from max bin value 3. Neutral – Points from zero

Saving/Loading Models

```
> saveRDS(mod, “my_model1.rds”)
> mod <- readRDS(“my_model1.rds”)
> save(mod, “old_way_of_doing_it.rda”) # not recommended
```

Binnr Ecosystem

There are currently two packages that enhance binnr in various ways. The first, binnrtools, provides additional functionality for binnr such as reporting tools. The second, mkivtools, allows the modeler to register a mkiv. If binnr detects a registered mkiv, additional functionality is enabled.

Binnrtools

Binnrtools provides access to a layout manager object that provides easy exporting of bin objects to Excel.

```
> library(binnrtools)

> lm <- export_classing(mod, “bivariates”)
> lm$add_worksheet(“second tab”)
> mod$select(“another model”)
> lm$export_classing(mod, “second tab”, inmodel.only=TRUE)
> lm$open_workbook()
```

Mkivtools

Mkivtools allows the user to register a mkiv with the R system by simply passing in the file location. If binnr detects a registered mkiv two new capabilities are enabled:

1. Pulling up mkiv code while using the `adjust` function
2. Including mkiv variables in the generated SAS code

```
> library(mkivtools)
> register_mkiv(“Z:/Resources/_MKIV/consumer-mkiv/mk_iv_5_2_1.sas”)
> mod$adjust()
```

Within the adjust interactive loop, using the `?` command will display the mkiv code for the current variable in the Rstudio viewer pane:

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

```
*#rv_S65_SSN_Deceased;
    attrib rv_S65_SSN_Deceased length = 3.;
    if not (truedid or (ssnlength > 0))
        then rv_S65_SSN_Deceased = .;
    else ...
```

The other feature that is enabled when binnr detects a registered mkiv is automatic inclusion of mkiv variables during code generation. No new syntax is required. Please refer to `Generating SAS Code` for details.

Understanding Generated SAS Code

The first section of SAS code contains a set of macro assignment statements. This is where Adverse Action code defaults should be assigned for each variable.

```
/** Adverse Action Code Mappings **/
%let mod1_AA_01 = ""; /** iv_add_apt **/
%let mod1_AA_02 = ""; /** rv_S65_SSN_Deceased **/
%let mod1_AA_03 = ""; /** rv_S65_SSN_Invalid **/
%let mod1_AA_04 = ""; /** rv_S65_SSN_Problem **/
%let mod1_AA_05 = ""; /** nf_age_at_ssn_issuance **/
%let mod1_AA_06 = ""; /** rv_P88_Phn_Dst_to_Inp_Add **/
```

The code for each model variable is printed one after the other in three parts. The first contains the weight-of-evidence substitution.

```
/*** iv_add_apt ***/
if missing(iv_add_apt)
  then mod1_V01_w = 0;
else if iv_add_apt <= 0.5
  then mod1_V01_w = 0.0917193779643158;
else mod1_V01_w = -0.288598842514218;
```

The second maps each bin level to an adverse action code. If none is provided, the default supplied in step one is used. Finally, the Adverse Action code distance is calculated from the requested reference point (min, max, neutral)

```
if missing(iv_add_apt)
  then mod1_AA_code_01 = "&mod1_AA_01";
else if iv_add_apt <= 0.5
  then mod1_AA_code_01 = "&mod1_AA_01";
else mod1_AA_code_01 = "&mod1_AA_01";

mod1_AA_dist_01 = -0.288598842514218 - mod1_V01_w;
```

After all of the WoE and AA assignments, the piece of generated code is the model equation summing each individual variable contribution along with the intercept. Subsequent scaling should reference `pfx_lgt` which is the predicted log-odds of the model.

```
/*** Final Score Calculation ***/
mod1_lgt = -2.63505173770652
  + mod1_V01_w
  + mod1_V02_w
  + mod1_V03_w
  + mod1_V04_w
  + mod1_V05_w
  + mod1_V06_w
;
```