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 monotoncity • 1 Increasing • 2 Either increasing or decreasing
exceptions	Values to withhold from discretization

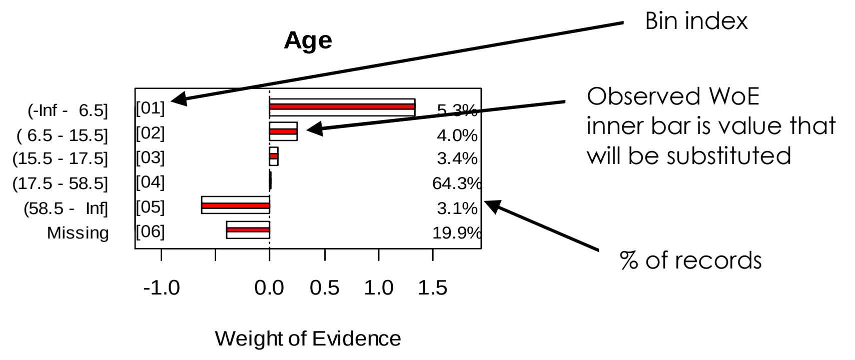
Handling Multi-collinearity

```
> cc <- mod$cluster()
> to_drop <- mod$prune_clusters(cc, corr=0.90, 1)
> mod$drop(to_drop)
```

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$select("model 1") # select any model that has been fitted
> mod$sort() # sort by inmodel, not dropped, then IV
> mod$summary() # print summary statistics
> mod$compare("model 1", "model 2") # compare coefs & contrib
```

Groups of Variables

```
> mod$get_dropped(invert = ) # optionally invert selection
[1] "Survived"
> mod$get_inmodel(invert = ) # optionally invert selection
[1] "Pclass"   "Sex"      "Age"      "Fare"     "Embarked"
> mod$drop(c("Pclass", "Fare"))
> mod$get_dropped()
[1] "Survived" "Pclass"   "Fare"
```

Adjusting Bins

```
mod$adjust()
```

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. This is very different from how previous versions of `binnr` worked. This means you no longer have to assign the return value.

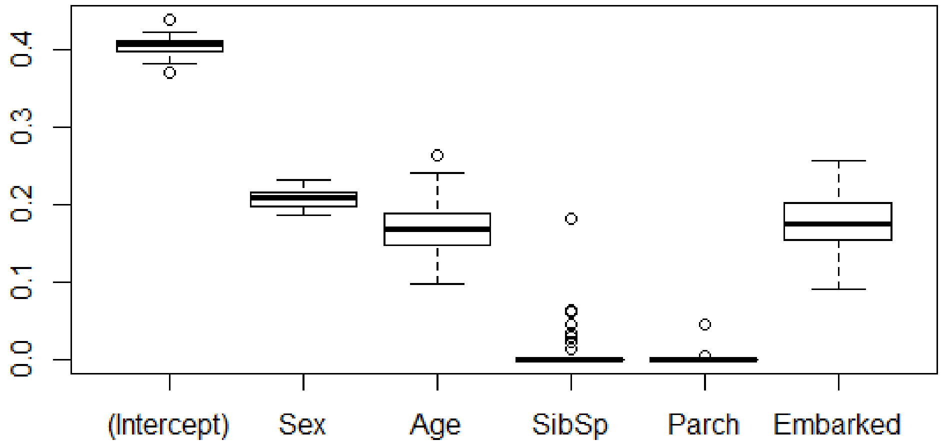
Pseudo P-Values

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.

```
> p <- mod$predict()
> p[1:4]
[1] -2.3726886  2.7146116  0.2156492  2.0621833

> 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(head(code, 17), sep="\n", file=”my_sas_code.sas”)

/** Adverse Action Code Mappings **/
%let mod1_AA_01 = ""; /** Pclass **/

/*** Pclass ***/
if missing(Pclass)
  then mod1_V01_w = 0;
else if Pclass in ('1')
  then mod1_V01_w = 1.06481036755641;
else if Pclass in ('2')
  then mod1_V01_w = 0.386593358951506;
else if Pclass in ('3')
  then mod1_V01_w = -0.706909416841032;
else mod1_V01_w = 0;

if missing(Pclass)
  then mod1_AA_code_01 = "&mod1_AA_01";
else if Pclass in ('1')
  then mod1_AA_code_01 = "&mod1_AA_01";
else if Pclass in ('2')
  then mod1_AA_code_01 = "&mod1_AA_01";
else if Pclass in ('3')
  then mod1_AA_code_01 = "&mod1_AA_01";
else mod1_AA_code_01 = "&mod1_AA_01";
```

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