GameRank Vignette

# Introduction

This vignette describes how to perform wrapper-based feature selection using the GameRank package. In feature selection scenario the likely following individual steps will be performed

1. Feature screening - Evaluating how much information each feature contains about the outcome
2. Feature construction - If the data doesn’t comprise good features, try to construct better ones
3. Feature selection - Apply variable selection methods to determine best combination, here: we’ll use wrappers
4. Model evaluation - Check performance of final model on hold-out data
5. Model exploitation - Using the model [not discussed here]

Within this vignette, we’ll use the following toy dataset

summary( toy\_data )

## USUBJID dys srv resp pi reg the\_normal the\_squared the\_cubed   
## Length:360 Min. : 0.0 Min. : 0.000 Mode :logical Min. :0.01522 Min. :-4.1698 Min. :-2.3946 Min. :0.001092 Min. :-0.002901   
## Class :character 1st Qu.: 60.5 1st Qu.: 2.000 FALSE:191 1st Qu.:0.28603 1st Qu.:-0.9148 1st Qu.: 0.1939 1st Qu.:0.115413 1st Qu.: 0.050081   
## Mode :character Median :121.0 Median : 4.000 TRUE :169 Median :0.46064 Median :-0.1578 Median : 0.9770 Median :0.233741 Median : 0.133504   
## Mean :148.6 Mean : 4.911 Mean :0.47470 Mean :-0.1320 Mean : 0.9292 Mean :0.294656 Mean : 0.198087   
## 3rd Qu.:211.8 3rd Qu.: 7.000 3rd Qu.:0.66193 3rd Qu.: 0.6719 3rd Qu.: 1.6482 3rd Qu.:0.432043 3rd Qu.: 0.255979   
## Max. :453.8 Max. :15.000 Max. :0.97135 Max. : 3.5234 Max. : 4.1593 Max. :1.078916 Max. : 1.507784   
## the\_exped the\_multi the\_power rnd01 rnd02 rnd03 rnd04 rnd05   
## Min. :0.5757 Min. :-0.1031 Min. :-1.376e-03 Min. :-0.88971 Min. :-0.81383 Min. :-0.8850087 Min. :-0.89564 Min. :-0.9322173   
## 1st Qu.:0.9779 1st Qu.: 0.1800 1st Qu.: 0.000e+00 1st Qu.:-0.25048 1st Qu.:-0.25494 1st Qu.:-0.2303074 1st Qu.:-0.21463 1st Qu.:-0.2060636   
## Median :1.1206 Median : 0.2907 Median : 1.084e-05 Median : 0.03073 Median :-0.02027 Median :-0.0007376 Median : 0.01041 Median : 0.0000143   
## Mean :1.1409 Mean : 0.4042 Mean : 1.037e-03 Mean : 0.00000 Mean : 0.00000 Mean : 0.0000000 Mean : 0.00000 Mean : 0.0000000   
## 3rd Qu.:1.2700 3rd Qu.: 0.6501 3rd Qu.: 4.243e-04 3rd Qu.: 0.23761 3rd Qu.: 0.22836 3rd Qu.: 0.2622294 3rd Qu.: 0.21514 3rd Qu.: 0.1930317   
## Max. :1.9995 Max. : 1.0621 Max. : 6.005e-02 Max. : 0.92190 Max. : 0.96105 Max. : 0.8657564 Max. : 0.83439 Max. : 0.9655277   
## rnd06 rnd07 rnd08 rnd09 rnd10 rnd11 rnd12 rnd13   
## Min. :-0.844021 Min. :-0.72838 Min. :-0.8440281 Min. :-0.85236 Min. :-0.69962 Min. :-0.861195 Min. :-0.7320347 Min. :-0.6957611   
## 1st Qu.:-0.214873 1st Qu.:-0.20140 1st Qu.:-0.1997045 1st Qu.:-0.21348 1st Qu.:-0.21359 1st Qu.:-0.193076 1st Qu.:-0.2108078 1st Qu.:-0.1630727   
## Median : 0.003165 Median :-0.00881 Median :-0.0006736 Median : 0.02299 Median : 0.01174 Median : 0.006763 Median :-0.0000712 Median : 0.0009211   
## Mean : 0.000000 Mean : 0.00000 Mean : 0.0000000 Mean : 0.00000 Mean : 0.00000 Mean : 0.000000 Mean : 0.0000000 Mean : 0.0000000   
## 3rd Qu.: 0.194128 3rd Qu.: 0.19739 3rd Qu.: 0.2039938 3rd Qu.: 0.19767 3rd Qu.: 0.21289 3rd Qu.: 0.198465 3rd Qu.: 0.1879815 3rd Qu.: 0.1554372   
## Max. : 0.974983 Max. : 1.00194 Max. : 0.7683129 Max. : 0.69115 Max. : 0.66944 Max. : 0.689377 Max. : 0.7994306 Max. : 0.7716363   
## rnd14 rnd15 rnd16 rnd17 rnd18 rnd19 rnd20   
## Min. :-0.75280 Min. :-0.84600 Min. :-0.69143 Min. :-0.673724 Min. :-0.760062 Min. :-0.595305 Min. :-0.6708420   
## 1st Qu.:-0.18224 1st Qu.:-0.17793 1st Qu.:-0.17509 1st Qu.:-0.146922 1st Qu.:-0.163002 1st Qu.:-0.147615 1st Qu.:-0.1360165   
## Median : 0.01635 Median :-0.02168 Median :-0.01638 Median :-0.001742 Median : 0.005179 Median : 0.002264 Median :-0.0008737   
## Mean : 0.00000 Mean : 0.00000 Mean : 0.00000 Mean : 0.000000 Mean : 0.000000 Mean : 0.000000 Mean : 0.0000000   
## 3rd Qu.: 0.18124 3rd Qu.: 0.16506 3rd Qu.: 0.17579 3rd Qu.: 0.160155 3rd Qu.: 0.168311 3rd Qu.: 0.168549 3rd Qu.: 0.1515780   
## Max. : 0.68423 Max. : 0.75145 Max. : 0.85242 Max. : 0.729226 Max. : 0.719546 Max. : 0.724555 Max. : 0.6211881

vars <- grep( "the\_|rnd", colnames(toy\_data), value=TRUE )  
resp <- "resp"

# 1. Feature screening

Let’s start with variable screening. This is to check for missing data, outliers and evaluate how much information the variables bear about the response variable.

GameRank provides a one-stop function for that: check\_variables function.

vck <- check\_variables( toy\_data, resp, vars )

## Evaluating variable resp   
## Evaluating variable the\_normal   
## Evaluating variable the\_squared   
## Evaluating variable the\_cubed   
## Evaluating variable the\_exped   
## Evaluating variable the\_multi   
## Evaluating variable the\_power

## Warning in KL.plugin(freqs2d, freqs.null, unit = unit): Vanishing value(s) in argument freqs2!

## Evaluating variable rnd01   
## Evaluating variable rnd02   
## Evaluating variable rnd03   
## Evaluating variable rnd04   
## Evaluating variable rnd05   
## Evaluating variable rnd06   
## Evaluating variable rnd07   
## Evaluating variable rnd08   
## Evaluating variable rnd09   
## Evaluating variable rnd10   
## Evaluating variable rnd11   
## Evaluating variable rnd12   
## Evaluating variable rnd13   
## Evaluating variable rnd14   
## Evaluating variable rnd15   
## Evaluating variable rnd16   
## Evaluating variable rnd17   
## Evaluating variable rnd18   
## Evaluating variable rnd19   
## Evaluating variable rnd20

vck %>% summary

## variable N n nmiss p check\_missing type entropy mutual\_information check\_entropy  
## Length:27 Min. :360 Min. :360 Min. :0 Min. :100 Drop : 0 Entropy not done: 0 Min. :0.6978 Min. :0.0004686 Entropy too low: 0   
## Class :character 1st Qu.:360 1st Qu.:360 1st Qu.:0 1st Qu.:100 Bad : 0 real :26 1st Qu.:1.2821 1st Qu.:0.0045804 Entropy ok :27   
## Mode :character Median :360 Median :360 Median :0 Median :100 Try : 0 integer : 0 Median :1.3613 Median :0.0086891   
## Mean :360 Mean :360 Mean :0 Mean :100 Good : 0 categorical : 0 Mean :1.4364 Mean :0.0154120   
## 3rd Qu.:360 3rd Qu.:360 3rd Qu.:0 3rd Qu.:100 Perfect:27 binary : 1 3rd Qu.:1.4941 3rd Qu.:0.0137805   
## Max. :360 Max. :360 Max. :0 Max. :100 Max. :3.1036 Max. :0.1579696   
## NA's :1   
## is\_response rot.nmin rot.nmax rot.p rng\_sd   
## Mode :logical Min. :0.000 Min. : 0.000 Min. : 0.0000 Min. : 4.145   
## FALSE:26 1st Qu.:0.000 1st Qu.: 0.000 1st Qu.: 0.2778 1st Qu.: 5.318   
## TRUE :1 Median :1.000 Median : 1.000 Median : 0.8333 Median : 5.586   
## Mean :1.154 Mean : 5.115 Mean : 1.7415 Mean : 5.840   
## 3rd Qu.:2.000 3rd Qu.: 2.750 3rd Qu.: 1.5278 3rd Qu.: 6.004   
## Max. :4.000 Max. :61.000 Max. :18.0556 Max. :12.880   
## NA's :1 NA's :1 NA's :1 NA's :1

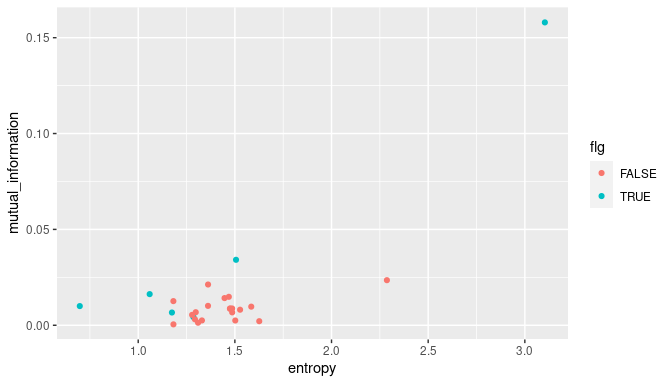
vck %>% filter( !is\_response ) %>% arrange( desc(entropy) )

## # A tibble: 26 × 15  
## variable N n nmiss p check\_missing type entropy mutual\_information check\_entropy is\_response rot.nmin rot.nmax rot.p rng\_sd  
## <chr> <int> <int> <int> <dbl> <fct> <fct> <dbl> <dbl> <fct> <lgl> <int> <int> <dbl> <dbl>  
## 1 the\_normal 360 360 0 100 Perfect real 3.10 0.158 Entropy ok FALSE 2 2 1.11 6.18  
## 2 rnd20 360 360 0 100 Perfect real 2.29 0.0235 Entropy ok FALSE 4 2 1.67 5.67  
## 3 rnd01 360 360 0 100 Perfect real 1.63 0.00212 Entropy ok FALSE 0 0 0 5.21  
## 4 rnd02 360 360 0 100 Perfect real 1.58 0.00971 Entropy ok FALSE 0 1 0.278 5.21  
## 5 rnd04 360 360 0 100 Perfect real 1.53 0.00811 Entropy ok FALSE 3 0 0.833 5.39  
## 6 the\_cubed 360 360 0 100 Perfect real 1.51 0.0342 Entropy ok FALSE 0 28 7.78 6.78  
## 7 rnd03 360 360 0 100 Perfect real 1.50 0.00248 Entropy ok FALSE 0 0 0 5.29  
## 8 rnd08 360 360 0 100 Perfect real 1.49 0.00672 Entropy ok FALSE 1 0 0.278 5.46  
## 9 rnd06 360 360 0 100 Perfect real 1.49 0.00875 Entropy ok FALSE 2 1 0.833 5.94  
## 10 rnd10 360 360 0 100 Perfect real 1.48 0.00873 Entropy ok FALSE 0 0 0 4.75  
## # … with 16 more rows

A look into the variable entropy and mutual information with respect to the response is a good idea to identify variables that are constant or bear low information

vck %>%   
 mutate( flg = grepl( "the\_", variable )) %>%  
 ggplot(aes(x=entropy, y=mutual\_information, color=flg ) ) +  
 geom\_point()

## Warning: Removed 1 rows containing missing values (geom\_point).



vck %>% arrange( desc(mutual\_information), desc(entropy) )

## # A tibble: 27 × 15  
## variable N n nmiss p check\_missing type entropy mutual\_information check\_entropy is\_response rot.nmin rot.nmax rot.p rng\_sd  
## <chr> <int> <int> <int> <dbl> <fct> <fct> <dbl> <dbl> <fct> <lgl> <int> <int> <dbl> <dbl>  
## 1 the\_normal 360 360 0 100 Perfect real 3.10 0.158 Entropy ok FALSE 2 2 1.11 6.18  
## 2 the\_cubed 360 360 0 100 Perfect real 1.51 0.0342 Entropy ok FALSE 0 28 7.78 6.78  
## 3 rnd20 360 360 0 100 Perfect real 2.29 0.0235 Entropy ok FALSE 4 2 1.67 5.67  
## 4 rnd11 360 360 0 100 Perfect real 1.36 0.0213 Entropy ok FALSE 2 0 0.556 5.54  
## 5 the\_power 360 360 0 100 Perfect real 1.06 0.0162 Entropy ok FALSE 4 61 18.1 12.9   
## 6 rnd05 360 360 0 100 Perfect real 1.47 0.0148 Entropy ok FALSE 3 1 1.11 6.11  
## 7 rnd07 360 360 0 100 Perfect real 1.45 0.0142 Entropy ok FALSE 0 3 0.833 5.75  
## 8 rnd18 360 360 0 100 Perfect real 1.18 0.0126 Entropy ok FALSE 1 2 0.833 6.13  
## 9 rnd13 360 360 0 100 Perfect real 1.36 0.0101 Entropy ok FALSE 2 5 1.94 5.48  
## 10 the\_squared 360 360 0 100 Perfect real 0.698 0.0100 Entropy ok FALSE 0 8 2.22 4.61  
## # … with 17 more rows

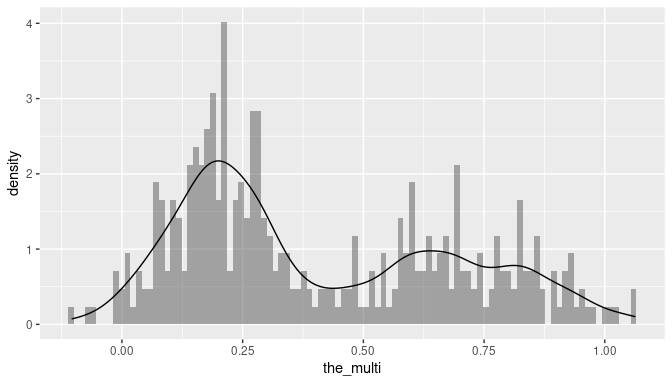
Sometimes variable distributions may be multi-modal. GameRank provides a function for this task: check\_multimodality.

GameRank determines multi-modal variables as follows: Let k be the number of mixture components ranging from 1 to k. The algorithm fits first m\_fits GMM models with k components using flexmix. Only models that converge are retained. For each k the minimum AIC is determined together with the number of converging models. All k with less than min\_fits\_converged models are removed. The k for which the minimum AIC is attained is then chosen. In case of ties for this AIC the minimum number of components are chosen. The first model obtaining these k and AIC is then used to determine cut-points if it has more than one component.

Cut-points are determined by the standard root finding algorithm, determining the points where the adjacent component distributions, scaled by their priors, are equal.

The chance for detecting multi-modal distributions depends on the available data and hence distributions reported may not be multi-modal or multi-modal distributions may go undetected. Thus a additional visual review of distributions is certainly a good idea, and the results from multi\_modal may be used as a starting point.

toy\_data %>%  
 ggplot( aes( x=the\_multi, y=..density.. ) ) +  
 geom\_histogram( bins = 100, alpha=0.5 ) +  
 geom\_density( bw = "ucv" )



mumo <- check\_multimodality( dat = toy\_data, resp = resp, vars = vars[1:9],n\_comp = 3, m\_fits = 25, min\_fits\_converged = 20 )

## Processing the\_normal   
## Processing the\_squared   
## Variable the\_squared is multi-modal with 2 Normal components. Determining cut-points.   
## Processing the\_cubed   
## Variable the\_cubed is multi-modal with 2 Normal components. Determining cut-points.   
## Processing the\_exped   
## Processing the\_multi   
## Variable the\_multi is multi-modal with 2 Normal components. Determining cut-points.   
## Processing the\_power   
## Variable the\_power is multi-modal with 3 Normal components. Determining cut-points.   
## Processing rnd01   
## Processing rnd02   
## Processing rnd03

mumo\_vars <- mumo$transforms %>% keep( function(x) !is.null( pluck( x, "transformed\_var" ) ) ) %>% map( "transformed\_var" ) %>% as.character  
mumo$data[,mumo\_vars]

## # A tibble: 360 × 4  
## the\_squared\_grp the\_cubed\_grp the\_multi\_grp the\_power\_grp  
## <chr> <chr> <chr> <chr>   
## 1 group[1] group[2] group[1] group[1]   
## 2 group[1] group[1] group[2] group[3]   
## 3 group[1] group[1] group[2] group[1]   
## 4 group[1] group[1] group[1] group[3]   
## 5 group[1] group[2] group[2] group[3]   
## 6 group[2] group[1] group[1] group[2]   
## 7 group[1] group[1] group[2] group[1]   
## 8 group[1] group[1] group[1] group[2]   
## 9 group[1] group[1] group[1] group[3]   
## 10 group[2] group[2] group[1] group[1]   
## # … with 350 more rows

mumo$transforms$the\_multi$aic\_aggregate

## # A tibble: 3 × 3  
## k min\_aic sum\_converged  
## <int> <dbl> <int>  
## 1 1 111. 25  
## 2 2 -34.8 25  
## 3 3 -28.8 2

mumo$transforms$the\_multi$best\_model

##   
## Call:  
## flexmix::flexmix(formula = stats::formula(sprintf("%s ~ 1", var)), data = dat[idx, ], k = k)  
##   
## Cluster sizes:  
## 1 2   
## 149 211   
##   
## convergence after 45 iterations

flexmix::parameters(mumo$transforms$the\_multi$best\_model)

## Comp.1 Comp.2  
## coef.(Intercept) 0.6967862 0.1932101  
## sigma 0.1630002 0.1014292

flexmix::prior(mumo$transforms$the\_multi$best\_model)

## [1] 0.4189274 0.5810726

pcuts <- mumo$transforms$the\_multi$cut\_points  
mumo$transforms$the\_multi$transformed\_var

## [1] "the\_multi\_grp"

Lets create categorical variable for the\_multi and add it to the set:

toy\_data <- toy\_data %>% bind\_cols( mumo$data[,mumo$transforms$the\_multi$transformed\_var] )  
vars <- c(vars, mumo$transforms$the\_multi$transformed\_var )

# 2. Feature construction

Another task may be to see if standard transforms improve Normality of the features. The one-stop function in GameRank tries square root, cube root, log and z-score transformations. Those that increase the Shapiro-Wilk W-statistics are retrained and added to the dataset.

smp <- simple\_transforms( toy\_data, vars = vars )

## Adding simple transformed variables if they are better by Shapiro-Wilk statistic W (larger is better)   
## Evaluating the\_normal with W = 0.9963

## Warning in FUN(X[[i]], ...): NaNs produced  
  
## Warning in FUN(X[[i]], ...): NaNs produced

## Evaluating the\_squared with W = 0.8973   
## Evaluating the\_cubed with W = 0.7652

## Warning in FUN(X[[i]], ...): NaNs produced  
  
## Warning in FUN(X[[i]], ...): NaNs produced

## Evaluating the\_exped with W = 0.9673   
## Evaluating the\_multi with W = 0.9272

## Warning in FUN(X[[i]], ...): NaNs produced  
  
## Warning in FUN(X[[i]], ...): NaNs produced

## Evaluating the\_power with W = 0.2100

## Warning in FUN(X[[i]], ...): NaNs produced  
  
## Warning in FUN(X[[i]], ...): NaNs produced

## Evaluating rnd01 with W = 0.9930

## Warning in FUN(X[[i]], ...): NaNs produced  
  
## Warning in FUN(X[[i]], ...): NaNs produced

## Evaluating rnd02 with W = 0.9944

## Warning in FUN(X[[i]], ...): NaNs produced  
  
## Warning in FUN(X[[i]], ...): NaNs produced

## Evaluating rnd03 with W = 0.9902

## Warning in FUN(X[[i]], ...): NaNs produced  
  
## Warning in FUN(X[[i]], ...): NaNs produced

## Evaluating rnd04 with W = 0.9961

## Warning in FUN(X[[i]], ...): NaNs produced  
  
## Warning in FUN(X[[i]], ...): NaNs produced

## Evaluating rnd05 with W = 0.9972

## Warning in FUN(X[[i]], ...): NaNs produced  
  
## Warning in FUN(X[[i]], ...): NaNs produced

## Evaluating rnd06 with W = 0.9978

## Warning in FUN(X[[i]], ...): NaNs produced  
  
## Warning in FUN(X[[i]], ...): NaNs produced

## Evaluating rnd07 with W = 0.9958

## Warning in FUN(X[[i]], ...): NaNs produced  
  
## Warning in FUN(X[[i]], ...): NaNs produced

## Evaluating rnd08 with W = 0.9974

## Warning in FUN(X[[i]], ...): NaNs produced  
  
## Warning in FUN(X[[i]], ...): NaNs produced

## Evaluating rnd09 with W = 0.9943

## Warning in FUN(X[[i]], ...): NaNs produced  
  
## Warning in FUN(X[[i]], ...): NaNs produced

## Evaluating rnd10 with W = 0.9928

## Warning in FUN(X[[i]], ...): NaNs produced  
  
## Warning in FUN(X[[i]], ...): NaNs produced

## Evaluating rnd11 with W = 0.9960

## Warning in FUN(X[[i]], ...): NaNs produced  
  
## Warning in FUN(X[[i]], ...): NaNs produced

## Evaluating rnd12 with W = 0.9966

## Warning in FUN(X[[i]], ...): NaNs produced  
  
## Warning in FUN(X[[i]], ...): NaNs produced

## Evaluating rnd13 with W = 0.9945

## Warning in FUN(X[[i]], ...): NaNs produced  
  
## Warning in FUN(X[[i]], ...): NaNs produced

## Evaluating rnd14 with W = 0.9959

## Warning in FUN(X[[i]], ...): NaNs produced  
  
## Warning in FUN(X[[i]], ...): NaNs produced

## Evaluating rnd15 with W = 0.9938

## Warning in FUN(X[[i]], ...): NaNs produced  
  
## Warning in FUN(X[[i]], ...): NaNs produced

## Evaluating rnd16 with W = 0.9966

## Warning in FUN(X[[i]], ...): NaNs produced  
  
## Warning in FUN(X[[i]], ...): NaNs produced

## Evaluating rnd17 with W = 0.9952

## Warning in FUN(X[[i]], ...): NaNs produced  
  
## Warning in FUN(X[[i]], ...): NaNs produced

## Evaluating rnd18 with W = 0.9977

## Warning in FUN(X[[i]], ...): NaNs produced  
  
## Warning in FUN(X[[i]], ...): NaNs produced

## Evaluating rnd19 with W = 0.9959

## Warning in FUN(X[[i]], ...): NaNs produced  
  
## Warning in FUN(X[[i]], ...): NaNs produced

## Evaluating rnd20 with W = 0.9964

## Warning in FUN(X[[i]], ...): NaNs produced  
  
## Warning in FUN(X[[i]], ...): NaNs produced

tfs <- smp$transformations %>% Reduce( bind\_rows, ., NULL )  
tfs %>% group\_by( variable ) %>% filter( max(W)==W )

## # A tibble: 26 × 5  
## # Groups: variable [26]  
## variable transformed\_var term W transform  
## <chr> <chr> <chr> <dbl> <chr>   
## 1 the\_normal "" "( the\_normal )" 0.996 identity   
## 2 the\_squared "the\_squared\_cubert" "( the\_squared )^(1/3)" 0.993 cube root  
## 3 the\_cubed "the\_cubed\_cubert" "( the\_cubed )^(1/3)" 0.994 cube root  
## 4 the\_exped "the\_exped\_log" " log( the\_exped ) " 0.995 log   
## 5 the\_multi "the\_multi\_cubert" "( the\_multi )^(1/3)" 0.963 cube root  
## 6 the\_power "the\_power\_log" " log( the\_power ) " 0.886 log   
## 7 rnd01 "" "( rnd01 )" 0.993 identity   
## 8 rnd02 "" "( rnd02 )" 0.994 identity   
## 9 rnd03 "" "( rnd03 )" 0.990 identity   
## 10 rnd04 "" "( rnd04 )" 0.996 identity   
## # … with 16 more rows

tfs %>% pull( transform ) %>% table

## .  
## cube root identity log sqrt zscore   
## 5 26 4 5 3

Lets add some transformed variables.

svars <- tfs %>% group\_by( variable ) %>% filter( max(W)==W ) %>% filter( "identity"!=transform )  
svars

## # A tibble: 5 × 5  
## # Groups: variable [5]  
## variable transformed\_var term W transform  
## <chr> <chr> <chr> <dbl> <chr>   
## 1 the\_squared the\_squared\_cubert "( the\_squared )^(1/3)" 0.993 cube root  
## 2 the\_cubed the\_cubed\_cubert "( the\_cubed )^(1/3)" 0.994 cube root  
## 3 the\_exped the\_exped\_log " log( the\_exped ) " 0.995 log   
## 4 the\_multi the\_multi\_cubert "( the\_multi )^(1/3)" 0.963 cube root  
## 5 the\_power the\_power\_log " log( the\_power ) " 0.886 log

toy\_data <- toy\_data %>% bind\_cols( smp$data[,svars$transformed\_var] )  
vars <- c(vars, svars$transformed\_var )

Another feature construction approach that can be tried in a second round is searching for Power-Transformations via the Box-Cox transformation. However we will skip this here. Please take a look at the example code for box\_cox\_normal and box\_cox\_binomial.

# 3. Feature selection

Now, let’s run two feature selection algorithms, the bidirectional search that applies forward and backward selection and the GameRank algorithm. First, we’ll split the dataset into thirds: one for training the model, one for validating it and one final hold-out dataset.

rr <- rep\_len( c(1L,2L,3L), length.out = nrow(toy\_data) )   
rr <- rr[ order( runif( length(rr) ) )]  
df\_test <- toy\_data[which(3==rr),]  
df\_sel <- toy\_data[which(rr %in% c(1L,2L)),]  
ds <- prepare\_splits( ds = 1L, dat = df\_sel, resp = resp, vars = vars, fn\_train = fn\_train\_binomial, fn\_eval = fn\_eval\_binomial\_auroc )

## Generating 1 splits

Wrapper selection algorithms are slow combinatorial searches that are not guaranteed to find more than a local optimum. Their performance also depends - in some cases - on the ordering of input variables. Therefore it is a good idea to rerun each algorithm with varying ordering of features to obtain a varying selections that can be used to choose from.

Here we’ll sort variables by their the mutual information with regards to the response:

vck <- check\_variables( df\_sel, resp, vars )

## Evaluating variable resp   
## Evaluating variable the\_normal   
## Evaluating variable the\_squared   
## Evaluating variable the\_cubed   
## Evaluating variable the\_exped   
## Evaluating variable the\_multi   
## Evaluating variable the\_power

## Warning in KL.plugin(freqs2d, freqs.null, unit = unit): Vanishing value(s) in argument freqs2!

## Evaluating variable rnd01   
## Evaluating variable rnd02   
## Evaluating variable rnd03   
## Evaluating variable rnd04   
## Evaluating variable rnd05   
## Evaluating variable rnd06   
## Evaluating variable rnd07   
## Evaluating variable rnd08   
## Evaluating variable rnd09   
## Evaluating variable rnd10   
## Evaluating variable rnd11   
## Evaluating variable rnd12   
## Evaluating variable rnd13   
## Evaluating variable rnd14   
## Evaluating variable rnd15   
## Evaluating variable rnd16   
## Evaluating variable rnd17   
## Evaluating variable rnd18   
## Evaluating variable rnd19   
## Evaluating variable rnd20   
## Evaluating variable the\_multi\_grp   
## Evaluating variable the\_squared\_cubert   
## Evaluating variable the\_cubed\_cubert   
## Evaluating variable the\_exped\_log   
## Evaluating variable the\_multi\_cubert   
## Evaluating variable the\_power\_log

## Warning in KL.plugin(freqs2d, freqs.null, unit = unit): Vanishing value(s) in argument freqs2!

vars <- vck %>% filter( !is\_response) %>% arrange( desc(mutual\_information) ) %>% pull( variable )

Let’s run the first wrapper: bidirectional search, an algorithm that performs a forward and backward selection step per iteration and ensures that it converges to the same partition by constraining the search variables for the forward and backward steps.

bds <- bidirectional( dat = df\_sel, resp = resp, vars = vars, fn\_train = fn\_train\_binomial, fn\_eval = fn\_eval\_binomial\_auroc, m = 6L, ds = ds, maximize = TRUE )

## Warning in max(.data$mean\_validation, na.rm = TRUE): no non-missing arguments to max; returning -Inf  
  
## Warning in max(.data$mean\_validation, na.rm = TRUE): no non-missing arguments to max; returning -Inf  
  
## Warning in max(.data$mean\_validation, na.rm = TRUE): no non-missing arguments to max; returning -Inf  
  
## Warning in max(.data$mean\_validation, na.rm = TRUE): no non-missing arguments to max; returning -Inf  
  
## Warning in max(.data$mean\_validation, na.rm = TRUE): no non-missing arguments to max; returning -Inf  
  
## Warning in max(.data$mean\_validation, na.rm = TRUE): no non-missing arguments to max; returning -Inf  
  
## Warning in max(.data$mean\_validation, na.rm = TRUE): no non-missing arguments to max; returning -Inf

bds$variable\_selections

## [[1]]  
## [1] "1" "rnd02" "rnd04" "rnd11" "rnd16" "the\_cubed" "the\_exped" "the\_normal"  
##   
## [[2]]  
## [1] "1" "rnd03" "rnd04" "rnd11" "rnd16" "the\_cubed" "the\_exped" "the\_normal"

bds$agg\_results %>% arrange( desc(mean\_validation) )

## # A tibble: 399 × 10  
## ch\_selection added selection m mean\_train mean\_validation mean\_bias opt k removed  
## <chr> <chr> <list> <int> <dbl> <dbl> <dbl> <lgl> <dbl> <chr>   
## 1 1,rnd02,rnd04,rnd11,rnd16,the\_cubed,the\_exped,the\_normal rnd02 <chr [8]> 8 0.778 0.768 0.0101 TRUE 7 <NA>   
## 2 1,rnd03,rnd04,rnd11,rnd16,the\_cubed,the\_exped,the\_normal rnd03 <chr [8]> 8 0.776 0.768 0.00797 TRUE 7 <NA>   
## 3 1,rnd04,rnd05,rnd11,rnd16,the\_cubed,the\_exped,the\_normal rnd05 <chr [8]> 8 0.776 0.768 0.00782 FALSE 7 <NA>   
## 4 1,rnd04,rnd11,rnd16,the\_cubed,the\_exped,the\_normal,the\_power the\_power <chr [8]> 8 0.777 0.767 0.0105 FALSE 7 <NA>   
## 5 1,rnd04,rnd11,rnd16,the\_cubed,the\_exped,the\_normal rnd16 <chr [7]> 7 0.777 0.766 0.0106 TRUE 6 <NA>   
## 6 1,rnd04,rnd06,rnd11,rnd16,the\_cubed,the\_exped,the\_normal rnd06 <chr [8]> 8 0.779 0.766 0.0124 FALSE 7 <NA>   
## 7 1,rnd04,rnd11,rnd12,rnd16,the\_cubed,the\_exped,the\_normal rnd12 <chr [8]> 8 0.777 0.766 0.0117 FALSE 7 <NA>   
## 8 1,rnd04,rnd11,rnd13,rnd16,the\_cubed,the\_exped,the\_normal rnd13 <chr [8]> 8 0.774 0.766 0.00825 FALSE 7 <NA>   
## 9 1,rnd04,rnd11,rnd16,the\_cubed,the\_exped,the\_multi\_grp,the\_normal the\_multi\_grp <chr [8]> 8 0.776 0.766 0.0102 FALSE 7 <NA>   
## 10 1,rnd04,rnd11,rnd16,rnd20,the\_cubed,the\_exped,the\_normal rnd20 <chr [8]> 8 0.790 0.765 0.0253 FALSE 7 <NA>   
## # … with 389 more rows

bds$agg\_results %>% arrange( desc(mean\_validation) ) %>% filter( opt )

## # A tibble: 8 × 10  
## ch\_selection added selection m mean\_train mean\_validation mean\_bias opt k removed  
## <chr> <chr> <list> <int> <dbl> <dbl> <dbl> <lgl> <dbl> <chr>   
## 1 1,rnd02,rnd04,rnd11,rnd16,the\_cubed,the\_exped,the\_normal rnd02 <chr [8]> 8 0.778 0.768 0.0101 TRUE 7 <NA>   
## 2 1,rnd03,rnd04,rnd11,rnd16,the\_cubed,the\_exped,the\_normal rnd03 <chr [8]> 8 0.776 0.768 0.00797 TRUE 7 <NA>   
## 3 1,rnd04,rnd11,rnd16,the\_cubed,the\_exped,the\_normal rnd16 <chr [7]> 7 0.777 0.766 0.0106 TRUE 6 <NA>   
## 4 1,rnd04,rnd11,the\_cubed,the\_exped,the\_normal the\_exped <chr [6]> 6 0.768 0.757 0.0113 TRUE 5 <NA>   
## 5 1,rnd04,rnd11,the\_cubed,the\_normal rnd04 <chr [5]> 5 0.764 0.747 0.0171 TRUE 4 <NA>   
## 6 1,rnd11,the\_cubed,the\_normal rnd11 <chr [4]> 4 0.762 0.730 0.0319 TRUE 3 <NA>   
## 7 1,the\_cubed,the\_normal the\_cubed <chr [3]> 3 0.750 0.705 0.0449 TRUE 2 <NA>   
## 8 1,the\_normal the\_normal <chr [2]> 2 0.733 0.670 0.0637 TRUE 1 <NA>

Now let’s run GameRank. Since GameRank doesn’t use a validation set, the dsi parameter receives an index vector of 1s and 2s that is then repeated to the length of the dataset and thereby defines the relative proportions of training to validation split per round. In small sample feature selection scenarios it contains just 2s such that all data are put into the validation split where the fn\_eval function performs a bootstrap or cross-validation (see small sample example code for details).

gmr <- game\_rank( dat = df\_sel, resp = resp, vars = vars, fn\_train = fn\_train\_binomial, fn\_eval = fn\_eval\_binomial\_auroc, m = 6L, dsi = c(1L,2L), maximize = TRUE,   
 team\_size = 3L, rounds = 10L, min\_matches\_per\_var = 5L )

## Comparing variable selections (# matches 40)---   
## Iteration 1 of 40 -- (+) : (-) scored 9 : 1   
## Iteration 2 of 40 -- (+) : (-) scored 10 : 0   
## Iteration 3 of 40 -- (+) : (-) scored 10 : 0   
## Iteration 4 of 40 -- (+) : (-) scored 3 : 7   
## Iteration 5 of 40 -- (+) : (-) scored 2 : 8   
## Iteration 6 of 40 -- (+) : (-) scored 0 : 10   
## Iteration 7 of 40 -- (+) : (-) scored 0 : 10   
## Iteration 8 of 40 -- (+) : (-) scored 10 : 0   
## Iteration 9 of 40 -- (+) : (-) scored 0 : 0   
## Iteration 10 of 40 -- (+) : (-) scored 8 : 2   
## Iteration 11 of 40 -- (+) : (-) scored 8 : 2   
## Iteration 12 of 40 -- (+) : (-) scored 8 : 2   
## Iteration 13 of 40 -- (+) : (-) scored 10 : 0   
## Iteration 14 of 40 -- (+) : (-) scored 3 : 0   
## Iteration 15 of 40 -- (+) : (-) scored 10 : 0   
## Iteration 16 of 40 -- (+) : (-) scored 8 : 2   
## Iteration 17 of 40 -- (+) : (-) scored 8 : 2   
## Iteration 18 of 40 -- (+) : (-) scored 0 : 10   
## Iteration 19 of 40 -- (+) : (-) scored 0 : 10   
## Iteration 20 of 40 -- (+) : (-) scored 0 : 10   
## Iteration 21 of 40 -- (+) : (-) scored 0 : 10   
## Iteration 22 of 40 -- (+) : (-) scored 0 : 10   
## Iteration 23 of 40 -- (+) : (-) scored 10 : 0   
## Iteration 24 of 40 -- (+) : (-) scored 10 : 0   
## Iteration 25 of 40 -- (+) : (-) scored 9 : 1   
## Iteration 26 of 40 -- (+) : (-) scored 10 : 0   
## Iteration 27 of 40 -- (+) : (-) scored 6 : 4   
## Iteration 28 of 40 -- (+) : (-) scored 10 : 0   
## Iteration 29 of 40 -- (+) : (-) scored 0 : 10   
## Iteration 30 of 40 -- (+) : (-) scored 9 : 1   
## Iteration 31 of 40 -- (+) : (-) scored 0 : 10   
## Iteration 32 of 40 -- (+) : (-) scored 7 : 3   
## Iteration 33 of 40 -- (+) : (-) scored 0 : 10   
## Iteration 34 of 40 -- (+) : (-) scored 9 : 1   
## Iteration 35 of 40 -- (+) : (-) scored 8 : 2   
## Iteration 36 of 40 -- (+) : (-) scored 3 : 7   
## Iteration 37 of 40 -- (+) : (-) scored 1 : 9   
## Iteration 38 of 40 -- (+) : (-) scored 0 : 0   
## Iteration 39 of 40 -- (+) : (-) scored 10 : 0   
## Iteration 40 of 40 -- (+) : (-) scored 3 : 7   
## Optimizing maximum likelihood   
## Calculating score vector   
## Calculating Hessian matrix   
## Compiling results

gmr$variable\_ranking %>% as.data.frame

## variable vs vs.var selected  
## 1 rnd16 1.668822594 2.958341e+03 TRUE  
## 2 rnd13 1.627152785 9.763520e+00 TRUE  
## 3 rnd08 1.387426662 1.335122e+03 TRUE  
## 4 rnd20 0.845005722 1.172076e+03 TRUE  
## 5 rnd09 0.841516182 7.495884e+01 TRUE  
## 6 the\_exped 0.826636937 3.097945e+01 TRUE  
## 7 rnd18 0.817756325 3.738408e+01 TRUE  
## 8 rnd04 0.783210452 1.084863e+01 TRUE  
## 9 the\_power\_log 0.594172748 5.602270e+00 TRUE  
## 10 rnd06 0.593515940 2.395008e+00 TRUE  
## 11 the\_exped\_log 0.569442260 4.259314e+03 TRUE  
## 12 the\_normal 0.569385887 6.392469e+03 TRUE  
## 13 rnd15 0.566875638 5.300191e+03 TRUE  
## 14 the\_power 0.564888300 6.907764e+02 TRUE  
## 15 rnd11 0.557548077 1.936503e+05 TRUE  
## 16 the\_squared\_cubert 0.549151213 2.219706e+03 TRUE  
## 17 rnd02 0.514391649 1.515098e+01 TRUE  
## 18 rnd05 0.503078604 1.965952e+01 TRUE  
## 19 the\_multi 0.498676121 7.831461e+02 TRUE  
## 20 rnd07 0.254751232 6.281291e+01 TRUE  
## 21 the\_cubed 0.008825978 6.420595e+03 TRUE  
## 22 rnd19 0.008526130 1.434653e+03 TRUE  
## 23 rnd10 0.008512025 4.546871e+02 TRUE  
## 24 the\_multi\_grp 0.006437998 1.336266e+02 TRUE  
## 25 rnd12 0.006036619 1.155678e+02 TRUE  
## 26 rnd01 -0.001281864 3.238858e+04 FALSE  
## 27 rnd03 -0.010134998 7.988526e+02 FALSE  
## 28 the\_cubed\_cubert -0.049212075 4.762528e+01 FALSE  
## 29 the\_squared -0.055515254 9.718121e+00 FALSE  
## 30 rnd17 -0.325289901 1.368980e+01 FALSE  
## 31 the\_multi\_cubert -0.580917222 4.400903e+00 FALSE  
## 32 rnd14 -1.122919840 3.487007e+02 FALSE

gmr$game\_rank\_selection

## [1] "rnd16" "rnd13" "rnd08" "rnd20" "rnd09" "the\_exped"

# 4. Model evaluation

Having obtained a potential variable selection, the model needs to assessed for calibration, that is whether it’s predictions correlate with the observed outcome. This may be easy for regression problems, for probability predictions or survival predictions it involves estimating the observed distribution.

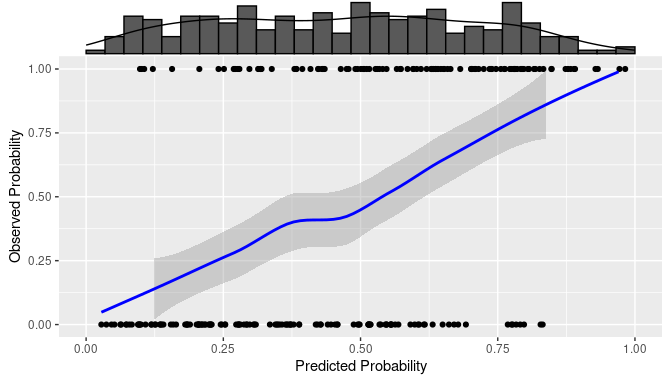
bds\_fsel <- bds %>% purrr::pluck( "variable\_selections" ) %>% purrr::pluck( 1L)  
mod\_bds <- fn\_train\_binomial( dat = df\_sel, resp = resp, selection = bds\_fsel )  
mod\_bds

##   
## Call: stats::glm(formula = fo, family = stats::binomial, data = dat)  
##   
## Coefficients:  
## (Intercept) rnd02 rnd04 rnd11 rnd16 the\_cubed the\_exped the\_normal   
## -2.6953 0.5421 0.7128 1.5355 -0.5153 2.2518 0.9866 0.9302   
##   
## Degrees of Freedom: 239 Total (i.e. Null); 232 Residual  
## Null Deviance: 331.6   
## Residual Deviance: 270 AIC: 286

gplot\_predictions\_binomial( dat = df\_sel, resp = resp, selection = bds\_fsel, mod = mod\_bds )

## `geom\_smooth()` using formula 'y ~ x'  
## `geom\_smooth()` using formula 'y ~ x'

## Warning: Removed 1 rows containing missing values (geom\_smooth).

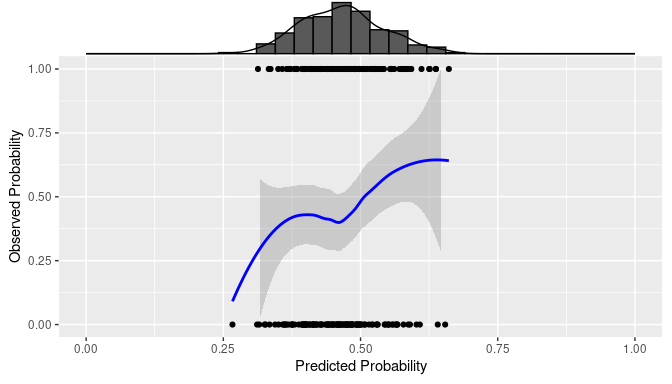


gmr\_fsel <- gmr %>% purrr::pluck( "game\_rank\_selection" )   
mod\_gmr <- fn\_train\_binomial( dat = df\_sel, resp = resp, selection = gmr\_fsel )  
mod\_gmr

##   
## Call: stats::glm(formula = fo, family = stats::binomial, data = dat)  
##   
## Coefficients:  
## (Intercept) rnd16 rnd13 rnd08 rnd20 rnd09 the\_exped   
## -0.96671 -0.07875 0.42836 0.09130 0.87538 0.42684 0.71406   
##   
## Degrees of Freedom: 239 Total (i.e. Null); 233 Residual  
## Null Deviance: 331.6   
## Residual Deviance: 326.3 AIC: 340.3

gplot\_predictions\_binomial( dat = df\_sel, resp = resp, selection = bds\_fsel, mod = mod\_gmr )

## `geom\_smooth()` using formula 'y ~ x'  
## `geom\_smooth()` using formula 'y ~ x'



To finally understand the model, it is a good idea to also identify influential observations that impact the model fit. With influential\_observations we can generate a list of observations that, if they are removed, reduce or increase a model parameter by more than Q1 - 1.5 x IQR and Q3 + 1.5 x IQR of the distribution of difference to the reference model.

ifo <- influential\_observations( df\_sel, resp, gmr\_fsel, fn\_train\_binomial, fn\_eval\_binomial\_auroc, fn\_infl\_coefficients, fn\_predict\_glm )  
ifo

## # A tibble: 241 × 14  
## row is\_influential is\_influential\_co ei deval yi dffit `(Intercept)\_dfbe… rnd16\_dfbeta rnd13\_dfbeta rnd08\_dfbeta rnd20\_dfbeta rnd09\_dfbeta  
## <int> <lgl> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 NA NA <NA> 0.582 NA NA NA -0.967 -0.0788 0.428 0.0913 0.875 0.427   
## 2 1 TRUE "(Intercept)\_dfbeta the\_expe… 0.582 0.000349 0.559 -0.0296 0.106 0.0640 -0.0160 0.0125 0.0463 0.0173   
## 3 2 FALSE "" 0.585 -0.00216 0.624 0.0138 -0.0214 -0.0209 0.0510 0.00685 0.0529 0.00561   
## 4 3 TRUE "rnd20\_dfbeta" 0.581 0.00133 0.488 -0.0391 0.0663 -0.0199 -0.0238 0.0464 0.111 -0.0144   
## 5 4 FALSE "" 0.584 -0.00202 0.532 -0.00950 0.0363 0.0468 0.0166 0.000935 -0.0158 0.00207   
## 6 5 FALSE "" 0.582 0.000558 0.522 0.00497 -0.0169 0.00862 0.000738 0.00549 0.0364 -0.00766   
## 7 6 FALSE "" 0.578 0.00439 0.425 -0.0194 -0.0401 -0.0425 -0.0594 -0.0215 0.0459 -0.000858  
## 8 7 FALSE "" 0.582 0.000419 0.601 0.0252 -0.0150 0.0200 0.0404 -0.0516 0.0255 0.0497   
## 9 8 TRUE "(Intercept)\_dfbeta the\_expe… 0.585 -0.00230 0.591 -0.0244 0.100 -0.0252 0.0111 -0.00534 0.0305 -0.0472   
## 10 9 FALSE "" 0.583 -0.000698 0.620 0.0173 -0.0342 -0.0299 0.0293 0.0457 0.0396 0.0211   
## # … with 231 more rows, and 1 more variable: the\_exped\_dfbeta <dbl>

ifo %>% filter( is\_influential )

## # A tibble: 29 × 14  
## row is\_influential is\_influential\_co ei deval yi dffit `(Intercept)\_df… rnd16\_dfbeta rnd13\_dfbeta rnd08\_dfbeta rnd20\_dfbeta rnd09\_dfbeta the\_exped\_dfbeta  
## <int> <lgl> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 1 TRUE (Intercept)\_dfbe… 0.582 3.49e-4 0.559 -0.0296 0.106 0.0640 -0.0160 0.0125 0.0463 0.0173 -0.0996   
## 2 3 TRUE rnd20\_dfbeta 0.581 1.33e-3 0.488 -0.0391 0.0663 -0.0199 -0.0238 0.0464 0.111 -0.0144 -0.0661   
## 3 8 TRUE (Intercept)\_dfbe… 0.585 -2.30e-3 0.591 -0.0244 0.100 -0.0252 0.0111 -0.00534 0.0305 -0.0472 -0.0954   
## 4 20 TRUE the\_exped\_dfbeta 0.583 -8.37e-4 0.580 -0.0222 0.0937 -0.0263 -0.0445 -0.0254 -0.00999 0.0344 -0.0891   
## 5 30 TRUE rnd09\_dfbeta 0.584 -1.74e-3 0.363 0.0161 0.0144 -0.0183 -0.000783 0.0193 -0.0279 -0.0783 -0.00359  
## 6 32 TRUE rnd20\_dfbeta 0.585 -2.16e-3 0.384 0.0310 0.0160 -0.0343 0.0851 0.0321 -0.117 0.00917 -0.00258  
## 7 38 TRUE rnd20\_dfbeta 0.579 3.07e-3 0.595 -0.0307 -0.0308 -0.0305 -0.0432 0.0734 -0.0951 -0.00610 0.0177   
## 8 54 TRUE (Intercept)\_dfbe… 0.581 1.26e-3 0.647 -0.0394 0.104 0.00883 -0.0447 -0.103 0.0137 -0.0513 -0.0995   
## 9 61 TRUE rnd09\_dfbeta 0.581 1.40e-3 0.370 0.0251 -0.0446 -0.00501 -0.0286 0.0451 -0.0406 -0.0813 0.0468   
## 10 68 TRUE rnd20\_dfbeta 0.584 -2.02e-3 0.309 0.0272 0.0329 0.0473 0.0178 0.0582 -0.0907 -0.0668 -0.0173   
## # … with 19 more rows