GameRank

Carsten Henneges

Joseph N Paulson

2022-01-07 17:34:47

# Introduction to GameRank

GameRank is a package for feature selection. With the advent of omics technologies and large multi-variate and multi-level datasets comprising 1,000s of features, feature selection became a necessary and critical part of modern data analysis. Goals are three-fold: improving prediction performance of trained models by removing irrelevant and noisy features (predictive modelling), obtaining sparse and parsymonious models (descriptive modelling), and - hopefully - gaining more insights on explaining the data generating process (explanatory modelling). [Shmueli and others (2010); Sauerbrei2020] This package and its vignette consider predominantly the predictive but also the descriptive goal while sparing out the explanatory modelling aspect, which first relies on a domain theory and a theoretical model that can be - at best - informed but not be automatically generated by algorithmic feature selection methods.

This vignette describes how to perform wrapper-based feature selection using the GameRank package for predictive and possibly descriptive modelling. In such a feature selection scenario likely the following individual steps will be performed: (Shmueli and others 2010)

1. Feature screening - Evaluating missing data, outliers and how much information each variable contains about the outcome, that is evaluating mutual information and correlations with the outcome
2. Feature construction - Try to derive more informative or better correlated features from available variables through combinations and/or functional transformations
3. Feature selection - Apply wrapper-based feature selection by GameRank to obtain an optimized combination for the prediction model or description model
4. Model evaluation - Check performance of final model on hold-out data, measure its descriptive qualities on the full dataset

These steps may be followed by exploiting the model or deploying the model for usage. Model exploitation may include finding optima or best points to determine next. Deploying the model may be generating nomograms or uploading to a website.

Within this vignette, we’ll demonstrate the package funcationality using the following toy dataset (see code in inst/data-raw/toy\_data.R for details; it includes four special variables that were sampled from a univariate Normal distribution and then squared, cubed, exp() and power transformed [the\_squared,the\_cubed,the\_exped,the\_power], and one variable that is sampled from a multi-modal Gaussian Mixture Distributions with two components [the\_multi]):

vars <- grep( "the\_|rnd", colnames(toy\_data), value=TRUE ) # Define a list of variables for selection  
resp <- "resp" # Define the response variable (can be a formula term)  
summary( toy\_data %>% dplyr::select( all\_of( c( resp, vars[1:4] ) ) ) )

## resp the\_normal the\_squared the\_cubed the\_exped   
## Mode :logical Min. :-1.6818 Min. :0.0000045 Min. :-0.0001198 Min. :0.4424   
## FALSE:298 1st Qu.: 0.3713 1st Qu.:0.1268884 1st Qu.: 0.0419764 1st Qu.:0.9308   
## TRUE :62 Median : 1.0130 Median :0.2384235 Median : 0.1240778 Median :1.0956   
## Mean : 1.0217 Mean :0.2781601 Mean : 0.1955912 Mean :1.1100   
## 3rd Qu.: 1.7131 3rd Qu.:0.3761562 3rd Qu.: 0.2839108 3rd Qu.:1.2624   
## Max. : 4.9241 Max. :1.1915268 Max. : 1.0272744 Max. :1.9547

The split into training, validation and hold-out test set are done when we actually run the wrapper algorithms.

# 1. Feature screening

Before building a predictive or descriptive model, it’s good practice to describe the features. That normally includes looking at the completeness of variables (missing data) and distributional features (like distribution skew, multi-modality or outliers). Additionally it is of interst how much information the data contains about the outcome? Most successful prediction approaches start with such a step. (Guyon and Elisseeff 2003)

GameRank provides a one-stop function for that: check\_variables(). This function compiles the total number of observations, the percentage of non-missing observations and categorizes completeness into categories from Perfect (100% complete) to Drop (<70% completeness).

It calculates the variable entropy and mutual information with respect to the response variable, which can be a categorical, numeric or surival outcome. To this end, it applies histogram density estimation with binwidth chosen be leave-one-out cross-validation, and estimates the mutual information from the cross-tabulated counts. This overview information allows to screen, filter or rank for variables with low entropy (coding for single infrequent values and low correlated variables with the outcome). It classifies variables by entropy into those that can be dropped (entropy<0.001) and might be kept.

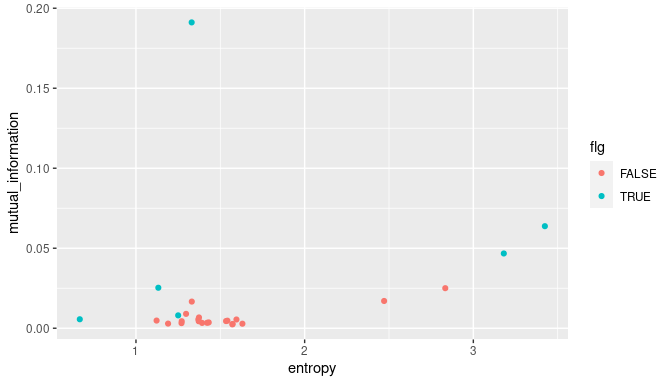
Finally, it determines, for numeric variables, as to whether there are outliers using the robust outlier test method together with the range-to-sd-ratio as measure for skew. Using this table, we can easily filter and screen for useful variables.

vck <- check\_variables( toy\_data, resp, vars )  
vck %>%   
 filter( !is\_response ) %>%   
 arrange( desc(entropy) ) %>%   
 dplyr::select( variable, p, check\_missing, entropy, mutual\_information, check\_entropy ) %>%  
 head

## # A tibble: 6 × 6  
## variable p check\_missing entropy mutual\_information check\_entropy  
## <chr> <dbl> <fct> <dbl> <dbl> <fct>   
## 1 the\_cubed 100 Perfect 3.42 0.0638 Entropy ok   
## 2 the\_multi 100 Perfect 3.18 0.0467 Entropy ok   
## 3 rnd02 100 Perfect 2.83 0.0250 Entropy ok   
## 4 rnd16 100 Perfect 2.47 0.0171 Entropy ok   
## 5 rnd01 100 Perfect 1.63 0.00284 Entropy ok   
## 6 rnd03 100 Perfect 1.60 0.00547 Entropy ok

Ranking and plotting variables by their entropy and mutual information with respect to the response helps to identify variables that are constant or contain low information

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



Sometimes variable distributions may be multi-modal, that is they may have more than one mode (point of probability mass concentration). Multiple modes indicate that the data for a variable may come from multiple data generating processes, that should be adjusted or better stratified for in subsequent analyses.

GameRank provides a function for this task: check\_multimodality. This function determines multi-modality for continuous numeric variables as follows:

for k from 1 to kmax do  
 - m <- list()  
 - for i from 1 to m\_fits od  
 - l <- list()  
 - fit Gaussian Mixture Model g with k components  
 - if is\_converged(g) then add g to list l fi  
 od  
od  
- Find the minimal AIC for which at least min\_fits\_converged\_models were recorded and  
 retrieve that model as gbest. In case of ties use the one with minimal k.  
if 1 < n\_components(gbest) then   
 - determine cut-points between mixture components via root finding algorithm  
fi

Cut-points are determined by a standard root finding algorithm, locating - if possible - 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 may go undetected. Thus a additional visual review of the identified variable distributions is a good idea, for which the results from multi\_modal may be used as a starting point.

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

The result of check\_modality consists of two list elements: “data” and “transforms”. The first, “data”, is a copy of the input data with added feature columns, in case a variable was found to be multi-modal. The “transforms” element is a list that comprises details for multi-modal distributions about the algorithm results, i.e. an AIC table and the optimal flexmix model, for instance.

To compile a handy table out of this list, we are using purrr to extract all non-null “transformed\_var” slots as a character list. Those contain the names of the newly generated multi-modal variables.

mumo\_vars <- mumo$transforms %>%  
 keep( ~ !is.null( .x$transformed\_var ) ) %>%   
 map\_chr( "transformed\_var" )   
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[2] group[2] group[1] group[1]   
## 2 group[1] group[2] group[2] group[1]   
## 3 group[1] group[1] group[2] group[2]   
## 4 group[2] group[3] group[2] group[2]   
## 5 group[2] group[3] group[1] group[1]   
## 6 group[1] group[3] group[2] group[1]   
## 7 group[1] group[3] group[1] group[1]   
## 8 group[2] group[3] group[1] group[1]   
## 9 group[1] group[2] group[1] group[2]   
## 10 group[2] group[2] group[1] group[1]   
## # … with 350 more rows

Every entry in the transforms list contains an AIC table with the model selection data

mumo$transforms$the\_multi$aic\_aggregate

## # A tibble: 3 × 3  
## k min\_aic sum\_converged  
## <int> <dbl> <int>  
## 1 1 81.6 25  
## 2 2 -80.1 25  
## 3 3 -80.1 1

The best GMM model from flexmix is saved as well. Using the parameters() and prior() functions we can obtain the differing component distribution parameters (mean and standard deviation) and the prior values (ie, how likely each mixture is).

With parameters from the flexmix package, we get a table where each column reports the mean and variance estimates of the mixture components. With prior we’ll get the mixing probability estimates.

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   
## 142 218   
##   
## convergence after 43 iterations

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

## Comp.1 Comp.2  
## coef.(Intercept) 0.6895635 0.20336168  
## sigma 0.1627790 0.09457608

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

## [1] 0.3995778 0.6004222

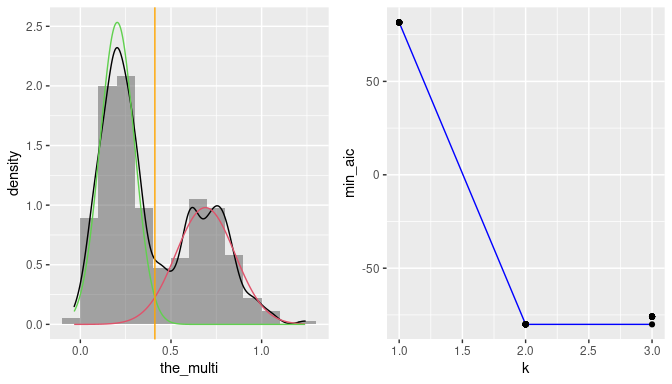
We can obtain the cut-points separating both modes:

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

## [1] -Inf 0.4111334 Inf

We can now display the multi-modal variable with the function gplot\_mulitmodal\_variable(). It displays a histogram and kernel density estimators overlayed by the best Gaussian Mixture distribution model components, and on the right side an Akaike Information Criterion plot with the line connecting those minimal AICs for which enough models converged:

gplot\_multimodal\_variable( dat = toy\_data, resp = resp, vars = vars[1:9],   
 mumo = mumo, variable = "the\_multi" )



# 2. Feature construction

Another task in predictive modeling is finding variables that are better when they are transformed. (Sauerbrei et al. 2020) refer to this as function selection, (Guyon and Elisseeff 2003) as feature construction. While feature selection is an algorithmic area, **feature construction** is less automated and requires domain knowledge. There is no unique recipe for constructing better features than trial and error.(Guyon and Elisseeff 2003) Hence analysts usually try a list of standard transforms (e.g., square root, log, some powers) to see if the distribution shapes comes closer to Normal, for instance.(Mangiafico 2016) But also Principal Component Analysis, Independent Component Analysis, Non-negative Matrix Factorization methods can be regarded as feature construction algorithms. Similarly, separating multiple modes - as done before - is a form of feature construction.

In the remainder of the last section, we observed a multi-modal variable. As cut-points are determined by check\_multimodality in those cases, we can easily add a new feature to our dataset that helps to distinguish between the component distributions:

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

GameRank also provides a handy function to check if standard transforms improve Normality of the features.(Mangiafico 2016) The one-stop function simple\_transforms in GameRank transforms each variable in the list by a square root, cube root, log and z-score transformation. Those that improve the Shapiro-Wilk W-statistics are then retrained and added to the dataset:

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

To see which variables were found, let’s shape the output into a tibble:

tfs <- purrr::map\_dfr(smp$transformations, ~ . )

Let’s filter and see which transforms gave the best improvements in Normality:

tfs %>% group\_by( variable ) %>% filter( max(W)==W ) %>% head

## # A tibble: 6 × 5  
## # Groups: variable [6]  
## variable transformed\_var term W transform  
## <chr> <chr> <chr> <dbl> <chr>   
## 1 the\_normal "" "( the\_normal )" 0.996 identity   
## 2 the\_squared "the\_squared\_sqrt" "sqrt( the\_squared )" 0.994 sqrt   
## 3 the\_cubed "the\_cubed\_cubert" "( the\_cubed )^(1/3)" 0.991 cube root  
## 4 the\_exped "the\_exped\_cubert" "( the\_exped )^(1/3)" 0.996 cube root  
## 5 the\_multi "the\_multi\_cubert" "( the\_multi )^(1/3)" 0.968 cube root  
## 6 the\_power "the\_power\_log" " log( the\_power ) " 0.849 log

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

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

We can add some further transformed variables to our dataset which we’ll next use for feature selection:

svars <- tfs %>%  
 group\_by( variable ) %>%   
 filter( max(W)==W ) %>%   
 filter( "identity"!=transform )  
toy\_data <- toy\_data %>% bind\_cols( smp$data[,svars$transformed\_var] )  
vars <- c(vars, svars$transformed\_var )

Other feature construction approaches are available through GameRank, like Power-Transformations via the Box-Cox transformation. Example code for those can be found at box\_cox\_normal and box\_cox\_binomial.

# 3. Feature selection

Feature selection can be done via three algorithmic approaches: (Guyon and Elisseeff 2003)

1. Filters that estimate and apply statistical quantities, like the correlation or mutual information, to select the best features;
2. Embedded methods that incorporate feature selection into model inference, like decision trees or gradient boosting methods; and
3. Wrapper methods that make use of a model training algorithm as a black-box to do a combinatorial search in the feature space, evaluating each selected combination via an user-defined loss function.

The third approach allows to directly optimize measures for calibration or discrimination and hence from a predictive modeling perspective seems to be desirable. Thus, let’s run two of those feature selection algorithms, the bidirectional search that applies forward and backward selection and the novel 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. We’ll generate and shuffle a vector or 1s, 2s and 3s which is as long as we have observations. Then we’ll separate rows with 3s (hold-off out).

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),] # Hold-out dataset  
df\_sel <- toy\_data[which(rr %in% c(1L,2L)),] # Data used for training and validation during wrapper selection

Next we’ll generate a matrix comprising 1s and 2s for training:validation splits. Each column denotes a separate split and the wrapper algorithm will use those defined splits **in parallel** for variable selection by averaging the validation performances. The function to do that is prepare\_splits taking the number ds of parallel-splits to generate, the dataset, list of response and features as well as the training and evaluation functions (fn\_train,fn\_eval) as arguments. It tries to find splits that are random but include a sufficient number of complete cases for model training.

ds <- prepare\_splits( ds = 3L, dat = df\_sel, resp = resp, vars = vars,   
 fn\_train = fn\_train\_binomial,   
 fn\_eval = fn\_eval\_binomial\_auroc )  
ds %>% head(5)

## [,1] [,2] [,3]  
## [1,] 1 1 1  
## [2,] 1 2 1  
## [3,] 1 2 2  
## [4,] 1 1 1  
## [5,] 2 2 2

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 shuffled orders of features to obtain a set of feature selections 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 )  
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 a backward selection step per iteration and ensures that all iterations converge to the same partition by constraining the searched 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 )  
bds$variable\_selections

## [[1]]  
## [1] "1" "rnd01" "rnd13" "rnd18" "the\_multi" "the\_normal"

bds$agg\_results %>%   
 as.data.frame %>%  
 arrange( desc(mean\_validation) ) %>%   
 filter( opt ) %>%  
 dplyr::select( ch\_selection, mean\_train, mean\_validation, mean\_bias ) %>%  
 head( 5 ) # Display top 5 optimal selections

## ch\_selection mean\_train mean\_validation mean\_bias  
## 1 1,rnd01,rnd13,rnd18,the\_multi,the\_normal 0.8600379 0.9086471 0.04860919  
## 2 1,rnd01,rnd03,rnd13,rnd18,the\_multi,the\_normal 0.8607312 0.9079861 0.04725496  
## 3 1,rnd01,rnd03,rnd13,rnd18,the\_multi,the\_normal,the\_power 0.8608008 0.9076797 0.04687895  
## 4 1,rnd01,rnd18,the\_multi,the\_normal 0.8593762 0.9058582 0.04648201  
## 5 1,rnd01,the\_multi,the\_normal 0.8602914 0.9045102 0.04421881

Let’s run GameRank next. GameRank doesn’t require the data to be split into training and validation sets. It receives a dsi parameter with an index vector of 1s and 2s that is then repeated up 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 dsi is set to 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(1,2) means 50:50 training:validation split each round;   
 # c(1,1,2) would be 2/3rds training to 1/3rds validation, and so forth  
 dsi = c(1L,2L),  
 maximize = TRUE,   
 team\_size = 4L, rounds = 10L, min\_matches\_per\_var = 7L )  
gmr$variable\_ranking %>%   
 head( 10 )

## # A tibble: 10 × 4  
## variable vs vs.var selected  
## <chr> <dbl> <dbl> <lgl>   
## 1 rnd14 1.26 12342. TRUE   
## 2 rnd17 1.25 23231754. TRUE   
## 3 rnd12 1.10 134476. TRUE   
## 4 rnd06 0.836 22688. TRUE   
## 5 the\_exped\_cubert 0.822 6010521. TRUE   
## 6 rnd11 0.733 17743. TRUE   
## 7 rnd03 0.636 16.2 TRUE   
## 8 rnd15 0.625 12147. TRUE   
## 9 rnd05 0.617 6.46 TRUE   
## 10 rnd04 0.458 806018. TRUE

gmr$game\_rank\_selection

## [1] "rnd14" "rnd17" "rnd12" "rnd06" "the\_exped\_cubert" "rnd11"

# 4. Model evaluation

Having selected an optimized feature set, the model prediction performance should be measured on the hold-out test dataset. A model is **calibrated** if it’s predictions match with the obersvations. That can be easily plotted for regression models, but involves estimating the unobserved probability distributions for binary and survival outcomes.

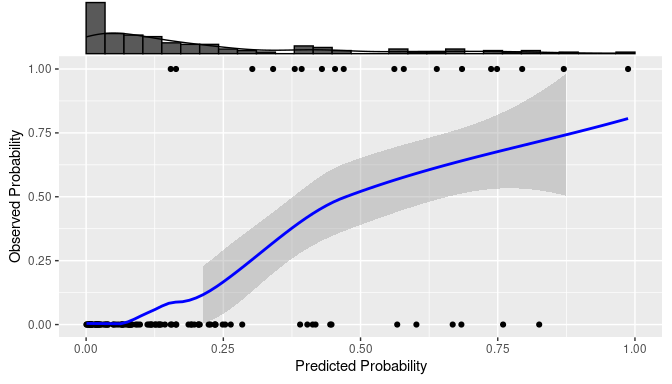
GameRank provides plotting functions for these cases (e.g., gplot\_predictions\_binomial and gplot\_predictions\_cox). These functions receive a dataset, response and feature lists, and a final model. Usually this finaly model, in this example obtained before from fn\_train\_binomial, is derived from training plus validation data and calibration is plotted on the hold-out test data.

Let plot this for both selections obtained from the bidirectional search and GameRank:

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) rnd01 rnd13 rnd18 the\_multi the\_normal   
## -4.8811 0.7983 -0.8411 0.2689 1.7503 1.8218   
##   
## Degrees of Freedom: 239 Total (i.e. Null); 234 Residual  
## Null Deviance: 228.7   
## Residual Deviance: 151.3 AIC: 163.3

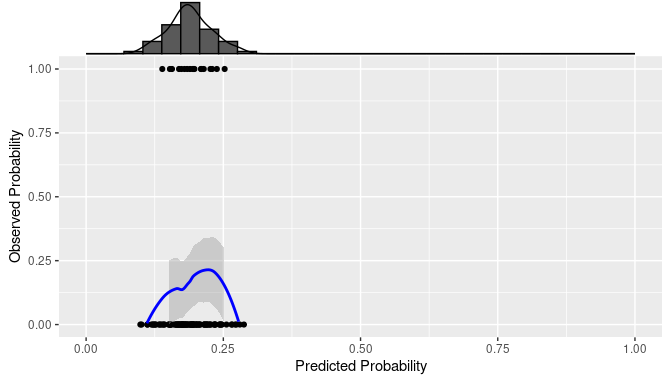
gplot\_predictions\_binomial( dat = df\_test, resp = resp,   
 selection = bds\_fsel, mod = mod\_bds )



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) rnd14 rnd17 rnd12 rnd06 the\_exped\_cubert rnd11   
## 1.70713 0.09341 -0.09121 0.45815 0.33450 -3.12184 -0.14868   
##   
## Degrees of Freedom: 239 Total (i.e. Null); 233 Residual  
## Null Deviance: 228.7   
## Residual Deviance: 225.5 AIC: 239.5

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



A last step is to understand the influence of observations on the predictive modeling. The goal is to identify influential observations that have impacted the model fit. GameRank identifies these influential\_observations as observations that, if they are removed, reduce or increase a model fit or any model parameter by more than and .

There is again a one-stop function influential\_observations the calculates those **dffits**- and **dfbeta**-like measures for each observation and flags observations accordingly:#’

ifo <- influential\_observations( df\_sel, resp, gmr\_fsel,   
 fn\_train\_binomial, fn\_eval\_binomial\_auroc,   
 fn\_infl\_coefficients, fn\_predict\_glm )  
ifo %>%   
 as.data.frame %>%   
 filter( is\_influential ) %>%  
 dplyr::select( row, is\_influential\_co, dffit ) %>%  
 head( 5 )

## row is\_influential\_co dffit  
## 1 2 dffit (Intercept)\_dfbeta rnd17\_dfbeta rnd12\_dfbeta the\_exped\_cubert\_dfbeta rnd11\_dfbeta 0.02848706  
## 2 3 deval dffit rnd14\_dfbeta rnd17\_dfbeta rnd06\_dfbeta 0.04221195  
## 3 15 deval dffit -0.01388222  
## 4 18 dffit -0.01823861  
## 5 21 deval dffit (Intercept)\_dfbeta rnd14\_dfbeta rnd12\_dfbeta rnd06\_dfbeta the\_exped\_cubert\_dfbeta rnd11\_dfbeta 0.03974594

# References

Guyon, Isabelle, and André Elisseeff. 2003. “An Introduction to Variable and Feature Selection.” *J. Mach. Learn. Res.* 3 (null): 1157–82.

Mangiafico, Salvatore S. 2016. “Transforming Data.”

Sauerbrei, Willi, Aris Perperoglou, Matthias Schmid, Michal Abrahamowicz, Heiko Becher, Harald Binder, Daniela Dunkler, Frank Harrell, Patrick Royston, and Georg Heinze. 2020. “State of the Art in Selection of Variables and Functional Forms in Multivariable Analysis—Outstanding Issues.” *Diagnostic and Prognostic Research* 4 (December). <https://doi.org/10.1186/s41512-020-00074-3>.

Shmueli, Galit, and others. 2010. “To Explain or to Predict?” *Statistical Science* 25 (3): 289–310.