# Reviewing the Example

A sales environment where we want to pick which prospects to target, using a model that predicts probability of conversion. We start with a data frame d of predicted probabilities and actual outcomes, and then determine the costs and rewards of different decisions. In our example, every contact costs $5 and every conversion brings a net revenue of $100. For demonstration purposes, we also add a small notional reward for true negatives and a small notional penalty for conversions that we missed.

# the data frame of model predictions and true outcome knitr::kable(head(d, n=3))

## converted predicted\_probability

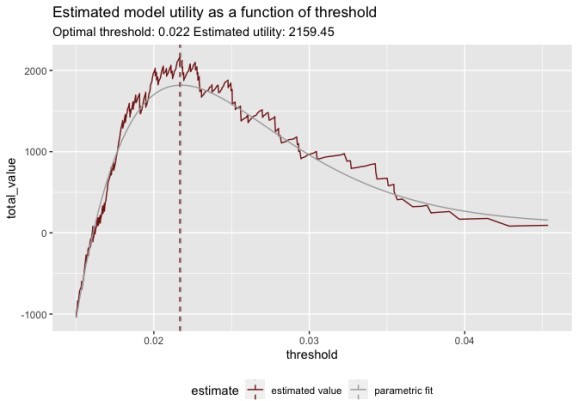
|  |  |
| --- | --- |
| FALSE | 0.0040164 |
| FALSE | 0.0199652 |
| FALSE | 0.0132867 |

# utilities

true\_positive\_value <- 100 - 5 # net revenue - cost false\_positive\_value <- -5 # the cost of a call true\_negative\_value <- 0.01 # a small reward for getting them right

false\_negative\_value <- -0.01 # a small penalty for having missed them

As we saw in the last article, this results in the following utility curve:



The best threshold is around 0.022, which realizes an estimated utility of $2159.45 on the evaluation set.

Since utility curves estimated on raw data sets can be noisy, it might be more stable to estimate the optimal threshold on a smoothed curve, like the one we’ve also plotted above. In this case, the optimal thresholds estimated on the raw data and on the smoothed curve are quite close (they match to two significant figures). Since the dollar amounts from the smoothed curve appears to be biased down in the region near the optimum, we’ll keep the estimate from the raw data. We’ll discuss the smoothing method used here a bit later, below.

# Estimating uncertainty bounds with bootstrapping

Now we have an optimal threshold, which we’ll call best\_threshold, and a estimate of the utility of that threshold. You might also want some uncertainty bars around that estimate. One way to estimate those uncertainty bars is with bootstrap sampling. In fact, if we are patient, we can simulate uncertainty bars for multiple thresholds (or the entire utility curve) simultaneously.

Bootstrap sampling simulates having multiple evaluation sets with similar characteristics by sampling from the original data set with replacement. We generate a new data set by resampling, then call sigr::model\_utility() on the new data to generate a new utility curve. Repeating the procedure over and over again produces a large collection of curves. These curves give us a distribution of plausible utility values for every threshold that we are interested in – in particular, best\_threshold. From this distribution, we can estimate uncertainty bounds.

We used the boot::boot() function to implement our bootstrapping. For simplicity, we use the raw percentiles from the replicants, and don’t attempt to correct for subsampling bias (which should be small for large data). We’ve wrapped the whole procedure in a function called estimate\_utility\_graph();This function generates 1000 bootstrap estimates from the original data, and returns the relevant summary statistics.

Calculate\_utility\_graph.R

|  |
| --- |
| # |
|  | # Runs a bootstrap estimation procedure to estimate uncertainty |
|  | # bounds on a given utility curve. |
|  | # |
|  | # Libraries used: sigr, boot, cdata, rquery/rqdatatable, wrapr |
|  | # |
|  | # Inputs: |
|  | # d: data frame of predictions, outcomes, and utilities |
|  | # 'true\_positive\_value': reward for a true positive (column name) |
|  | # 'false\_positive\_value': penalty for a false positive (column name) |
|  | # 'true\_negative\_value': reward for a true negative (column name) |
|  | # 'false\_negative\_value': penalty for a false negative (column name) |
|  | # prediction\_column\_name: name of predictions column |
|  | # outcome\_column\_name: name of outcome column |
|  | # |
|  | # Returns a list: |
|  | # plot\_thin: data frame in thin form of total value (utility) curve, and |
|  | # mean curve of bootstrap estimates; |
|  | # all as functions of threshold |
|  | # boot\_stats: frame of mean and median curves of bootstrap estimates, along with |
|  | # the boundary curves of the 95% and 50% quantiles, in wide form |
|  | # |
|  | estimate\_utility\_graph <- function( |
|  | d, |
|  | ..., |
|  | prediction\_column\_name, |
|  | outcome\_column\_name) { |
|  | wrapr::stop\_if\_dot\_args(substitute(list(...)), "estimate\_utility\_graph") |
|  |  |
|  | # calculate utility curve (actual values) |
|  | values <- model\_utility(d, prediction\_column\_name, outcome\_column\_name) |
|  |  |
|  | # get the thresholds |
|  | threshold\_list <- values$threshold[(!is.na(values$threshold))] |
|  |  |
|  | # calculates the utility curve from a bootstrap sample (described by indices) |
|  | f <- function(d, indices, ...) { |
|  | vi <- model\_utility(d[indices, ], prediction\_column\_name, outcome\_column\_name) |
|  | fn <- approxfun(vi$threshold, vi$total\_value, |
|  | yleft = min(vi$total\_value), |
|  | yright = max(vi$total\_value)) |
|  | fn(threshold\_list) |
|  | } |
|  |  |
|  | # run the bootstrap |
|  | boot\_stats <- boot(data = d, statistic = f, R = 1000, |
|  | parallel = 'multicore', ncpus = parallel::detectCores()) |
|  | boot\_data <- as.data.frame(boot\_stats$t) |
|  | # each column corresponds to a threshold |
|  | colnames(boot\_data) <- threshold\_list |
|  |  |
|  | # put it into long form |
|  | boot\_data <- pivot\_to\_blocks(boot\_data, |
|  | nameForNewKeyColumn = 'threshold', |
|  | nameForNewValueColumn = 'total\_value', |
|  | columnsToTakeFrom = colnames(boot\_data)) |
|  | # turn thresholds back into numbers |
|  | boot\_data$threshold <- as.numeric(boot\_data$threshold) |
|  |  |
|  | # functions to calculate quantiles |
|  | q\_0.025 <- function(x) { quantile(x, probs = 0.025) } |
|  | q\_0.25 <- function(x) { quantile(x, probs = 0.25) } |
|  | q\_0.50 <- function(x) { quantile(x, probs = 0.50) } |
|  | q\_0.75 <- function(x) { quantile(x, probs = 0.75) } |
|  | q\_0.975 <- function(x) { quantile(x, probs = 0.975) } |
|  |  |
|  | # create summary frame |
|  | boot\_summary <- project(boot\_data, |
|  | mean\_total\_value = mean(total\_value), |
|  | q\_0.025 = q\_0.025(total\_value), |
|  | q\_0.25 = q\_0.25(total\_value), |
|  | q\_0.50 = q\_0.50(total\_value), |
|  | q\_0.75 = q\_0.75(total\_value), |
|  | q\_0.975 = q\_0.975(total\_value), |
|  | groupby = 'threshold') %.>% |
|  | orderby(., 'threshold') |
|  |  |
|  | value\_range <- values[ |
|  | (values$threshold >= min(threshold\_list)) & |
|  | (values$threshold <= max(threshold\_list)), ] |
|  |  |
|  | # get the actual utility curve and the bootstrap mean curves |
|  | # (2 frames) |
|  | boot\_thin <- boot\_summary %.>% |
|  | select\_columns(., qc(threshold, mean\_total\_value)) %.>% |
|  | rename\_columns(., 'total\_value' := 'mean\_total\_value') %.>% |
|  | extend(., estimate = 'bootstrapped value') |
|  |  |
|  | value\_thin <- value\_range %.>% |
|  | select\_columns(., qc(threshold, total\_value)) %.>% |
|  | extend(., estimate = 'estimated value') |
|  | value\_thin <- value\_thin[complete.cases(value\_thin), , drop = FALSE] |
|  |  |
|  | plot\_thin <- rbind(boot\_thin, value\_thin) |
|  | plot\_thin <- plot\_thin[ |
|  | (complete.cases(plot\_thin)) & |
|  | (plot\_thin$threshold >= min(threshold\_list)) & |
|  | (plot\_thin$threshold <= max(threshold\_list)), ] |
|  |  |
|  | list(plot\_thin = plot\_thin, boot\_summary = boot\_summary) |
|  | } |
|  |  |
|  |  |
|  | # |
|  | # Estimate parametric ideal utility. |
|  | # |
|  | # Fit ideal beta curves and compute utility with respect to those curves |
|  | # |
|  | # Libraries used: sigr, cdata, rquery/rqdatatable, wrapr |
|  | # |
|  | # Inputs: |
|  | # d: data frame of predictions, outcomes, and utilities |
|  | # prediction\_column\_name: name of predictions column |
|  | # outcome\_column\_name: name of outcome column |
|  | # true\_positive\_value: reward for a true positive (scalar) |
|  | # false\_positive\_value: penalty for a false positive (scalar) |
|  | # true\_negative\_value: reward for a true negative (scalar) |
|  | # false\_negative\_value: penalty for a false negative (scalar) |
|  | # |
|  | # Returns a list: |
|  | # plot\_thin: data frame in thin form of total value (utility) curve, and |
|  | # mean curve of bootstrap estimates; |
|  | # all as functions of threshold |
|  | # boot\_stats: frame of mean and median curves of bootstrap estimates, along with |
|  | # the boundary curves of the 95% and 50% quantiles, in wide form |
|  | # |
|  | parametric\_utility\_graph <- function( |
|  | d, |
|  | ..., |
|  | prediction\_column\_name, |
|  | outcome\_column\_name, |
|  | true\_positive\_value, |
|  | false\_positive\_value, |
|  | true\_negative\_value, |
|  | false\_negative\_value) { |
|  | wrapr::stop\_if\_dot\_args(substitute(list(...)), "estimate\_utility\_graph") |
|  |  |
|  | # estimate a parametric value curve |
|  | # try to recover per-class beta distribution parameters |
|  | unpack[shape1\_pos, shape2\_pos, shape1\_neg, shape2\_neg] <- |
|  | sigr::find\_ROC\_matching\_ab(modelPredictions = d[[prediction\_column\_name]], |
|  | yValues = d[[outcome\_column\_name]]) |
|  | total\_pos <- sum(d[[outcome\_column\_name]]) |
|  | total\_neg <- sum(!d[[outcome\_column\_name]]) |
|  | # generate the required tails |
|  | theoretical\_values <- data.frame( |
|  | estimate = 'parametric fit', |
|  | threshold = values$threshold) |
|  | pos\_take\_count <- pbeta(theoretical\_values$threshold, shape1 = shape1\_pos, shape2 = shape2\_pos, lower.tail = FALSE) \* total\_pos |
|  | pos\_take\_count[is.na(pos\_take\_count)] <- 0 |
|  | neg\_take\_count <- pbeta(theoretical\_values$threshold, shape1 = shape1\_neg, shape2 = shape2\_neg, lower.tail = FALSE) \* total\_neg |
|  | neg\_take\_count[is.na(neg\_take\_count)] <- 0 |
|  | theoretical\_values$true\_negative\_count <- total\_neg - neg\_take\_count |
|  | theoretical\_values$false\_negative\_count <- total\_pos - pos\_take\_count |
|  | theoretical\_values$true\_positive\_count <- pos\_take\_count |
|  | theoretical\_values$false\_positive\_count <- neg\_take\_count |
|  | theoretical\_values$total\_value <- |
|  | theoretical\_values$true\_negative\_count \* true\_negative\_value + |
|  | theoretical\_values$false\_negative\_count \* false\_negative\_value + |
|  | theoretical\_values$true\_positive\_count \* true\_positive\_value + |
|  | theoretical\_values$false\_positive\_count \* false\_positive\_value |
|  | theoretical\_values |
|  | } |

An example use is as follows (we’ll restart from the beginning, so the code is all in one place):

library(wrapr) # misc convenience functions library(sigr) # for model\_utility() library(rquery) # data manipulation library(cdata) # data manipulation library(boot) # bootstrap sampling library source("calculate\_utility\_graph.R")

# d is the data frame of model predictions and outcomes # utilities

true\_positive\_value <- 100 - 5 # net revenue - cost false\_positive\_value <- -5 # the cost of a call true\_negative\_value <- 0.01 # a small reward for getting them right

false\_negative\_value <- -0.01 # a small penalty for having missed them

unpack[plot\_thin, boot\_summary] <- estimate\_utility\_graph( d,

prediction\_column\_name = "predicted\_probability", outcome\_column\_name = "converted", true\_positive\_value = true\_positive\_value, false\_positive\_value = false\_positive\_value, true\_negative\_value = true\_negative\_value, false\_negative\_value = false\_negative\_value)

The estimate\_utility\_graph() function returns two data frames, boot\_summary and plot\_thin. The data frame boot\_summary has columns for the mean utility curve over all the bootstrap samples, as well as curves for several key quantiles.

knitr::kable(head(boot\_summary, n=3))

|  |  |
| --- | --- |
| **threshold** | **mean\_total\_value q\_0.025 q\_0.25 q\_0.50 q\_0.75 q\_0.975** |
| 0.0002494 | -39415.70 -41500.00 -40100.00 -39500.00 -38700.00 -37400.00 |
| 0.0002564 | -39411.70 -41494.99 -40099.69 -39487.47 -38694.99 -37399.99 |
| 0.0002711 | -39407.39 -41489.98 -40093.71 -39479.96 -38689.98 -37394.86 |

One interesting uncertainty band is the range between the 2.5th percentile (q\_.0.025) and the 97.5th percentile (q\_0.975), which holds 95% of the observations. With some abuse of terminology, you can consider this analogous to a “95% confidence interval” for your estimated utility.

Right now, the function is hard coded to return estimates at all the same thresholds used in the original utility curve. You can use this data to get utility estimates at key threshold values, like best\_threshold.

# find the statistics corresponding to best\_threshold

ix = which(abs(boot\_summary$threshold - best\_threshold) < 1e-5)[1] best\_stats = boot\_summary[ix, ]

knitr::kable(best\_stats)

## threshold mean\_total\_value q\_0.025 q\_0.25 q\_0.50 q\_0.75 q\_0.975

9574 0.021672 2114.719 1020.626 1705.634 2083.686 2520.049 3379.423

At best\_threshold, we estimate that 95% of the time, the total utility realized will be in the range $1020.63 to $3379.42.

The plot\_thin data frame (in long form so it’s easy to use with ggplot2) again has the mean utility curve over all the bootstrap samples, the original utility curve from the real data, and the smoothed curve that we showed at the beginning of the article.

knitr::kable(head(plot\_thin, n=3))

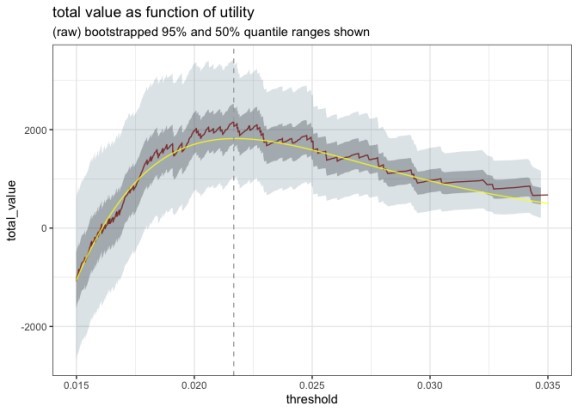
## threshold total\_value estimate

|  |  |
| --- | --- |
| 0.0002494 | -39415.70 bootstrapped value |
| 0.0002564 | -39411.70 bootstrapped value |
| 0.0002711 | -39407.39 bootstrapped value |

unique(plot\_thin$estimate)

## [1] "bootstrapped value" "estimated value" "parametric fit"

We can use these two data frames to plot the utility curve with uncertainty. Here we show the original curve, the smoothed curve, and the 50% and 95% quantile ranges around them.



**The smoothing curve**

Note that our estimate\_utility\_graph() function assumes that all rewards and costs are constant. This assumption isn’t necessary for the bootstrapping procedure; it’s only needed for the parametric smoothing curve that we calculate and add to the plot\_thin data frame.

Full Source Code

|  |
| --- |
|  |
| ```{r message=FALSE} |
|  | library(wrapr) |
|  | library(sigr) |
|  | library(rquery) |
|  | library(cdata) |
|  | library(ggplot2) |
|  | library(boot) |
|  | source("calculate\_utility\_graph.R") |
|  | ``` |
|  |  |
|  | ```{r} |
|  | # generate data (same example as ../UtilityExample.Rmd) |
|  | set.seed(2020) |
|  | y\_example <- function(n, prevalence = 0.5) { |
|  | data.frame( |
|  | y = sample( |
|  | c(TRUE, FALSE), |
|  | size = n, |
|  | replace = TRUE, |
|  | prob = c(prevalence, 1 - prevalence)) |
|  | ) |
|  | } |
|  | beta\_variable <- function( |
|  | d, |
|  | shape1\_pos, shape2\_pos, |
|  | shape1\_neg, shape2\_neg) { |
|  | score <- numeric(nrow(d)) |
|  | score[d$y] <- rbeta(sum(d$y), shape1 = shape1\_pos, shape2 = shape2\_pos) |
|  | score[!d$y] <- rbeta(sum(!d$y), shape1 = shape1\_neg, shape2 = shape2\_neg) |
|  | score |
|  | } |
|  | # |
|  | # generate the example. This looks complicated, but we are trying to generate |
|  | # predictions consistent with a calibrated model (like logistic regression) |
|  | # |
|  | prevalence <- 0.01 |
|  | a\_pos <- 6 |
|  | b\_pos <- 300 |
|  | a\_neg <- 3 |
|  | p\_odds\_ratio <- prevalence / (1-prevalence) |
|  | pos\_mean <- a\_pos / (a\_pos + b\_pos) |
|  | b\_neg <- a\_neg \* ( 1 / (p\_odds\_ratio \* (1 - pos\_mean)) - 1) |
|  | stopifnot(b\_neg >= 0) |
|  | neg\_mean <- a\_neg / (a\_neg + b\_neg) |
|  | check\_p <- prevalence \* pos\_mean + (1 - prevalence) \* neg\_mean |
|  | stopifnot(abs(prevalence - check\_p) < 1e-5) |
|  | d <- y\_example(10000, prevalence = prevalence) |
|  | d$predicted\_probability <- beta\_variable( |
|  | d, |
|  | shape1\_pos = a\_pos, |
|  | shape2\_pos = b\_pos, |
|  | shape1\_neg = a\_neg, |
|  | shape2\_neg = b\_neg) |
|  | # change the column name to better match the example |
|  | colnames(d) <- c("converted", "predicted\_probability") |
|  |  |
|  | ``` |
|  |  |
|  | We'll use the same example as in the previous article: a sales environment where we want to pick which prospects to target, |
|  | using a model that predicts probability of conversion. We start with a data frame `d` of predicted probabilities and actual outcomes, and then determine the costs and rewards of different decisions. In our example, every contact costs \$5 and every |
|  | conversion brings a net revenue of \$100. For demonstration purposes, we also add a small notional reward for true negatives and a small notional penalty for conversions that we missed. |
|  |  |
|  | ```{r echo=TRUE} |
|  | # the data frame of model predictions and true outcome |
|  | knitr::kable(head(d, n=3)) |
|  |  |
|  | # utilities |
|  | true\_positive\_value <- 100 - 5 # net revenue - cost |
|  | false\_positive\_value <- -5 # the cost of a call |
|  | true\_negative\_value <- 0.01 # a small reward for getting them right |
|  | false\_negative\_value <- -0.01 # a small penalty for having missed them |
|  | d$true\_positive\_value <- true\_positive\_value |
|  | d$false\_positive\_value <- false\_positive\_value |
|  | d$true\_negative\_value <- true\_negative\_value |
|  | d$false\_negative\_value <- false\_negative\_value |
|  |  |
|  | ``` |
|  |  |
|  |  |
|  | ```{r} |
|  | # estimate\_utility\_graph defined in |
|  | # https://github.com/WinVector/sigr/blob/main/extras/utility\_modeling/calculate\_utility\_graph.R |
|  |  |
|  | unpack[plot\_thin, boot\_summary] <- estimate\_utility\_graph( |
|  | d, |
|  | prediction\_column\_name = "predicted\_probability", |
|  | outcome\_column\_name = "converted") |
|  |  |
|  |  |
|  | ``` |
|  |  |
|  | As we saw in the last article, this results in the following utility curve: |
|  |  |
|  | ```{r} |
|  | # formatting |
|  | dollars\_cents = function(x) { |
|  | format(round(x, 2), nsmall=2) |
|  | } |
|  |  |
|  | # plot actual utility curve |
|  |  |
|  | unpack[bframe = "bootstrapped value", |
|  | eframe = "estimated value"] <- split(plot\_thin, plot\_thin$estimate) |
|  |  |
|  |  |
|  | get\_best\_threshold <- function(dframe) { |
|  | max\_ix <- which.max(dframe$total\_value) |
|  | best\_threshold <- dframe$threshold[max\_ix] |
|  | } |
|  |  |
|  | thresholds = data.frame(estimate = c(eframe$estimate[1]), |
|  | threshold = c(get\_best\_threshold(eframe))) |
|  |  |
|  | # use the threshold from the actual data |
|  | # in this case they are the same to 2 sig figs anyway |
|  | best\_threshold <- thresholds$threshold[1] |
|  | total\_utility <- max(eframe$total\_value) |
|  |  |
|  | subtitle = paste("Optimal threshold:", format(best\_threshold, digits=2), |
|  | "Estimated utility:", dollars\_cents(total\_utility)) |
|  |  |
|  |  |
|  |  |
|  | t\_bound <- 0.015 # only look at thresholds above t\_bound |
|  | pltframe <- subset(eframe, threshold >= t\_bound) |
|  |  |
|  | pal <- c('#841c17', 'darkgray') |
|  |  |
|  | ggplot() + |
|  | geom\_line(data=pltframe, |
|  | mapping=aes(x=threshold, y=total\_value, color=estimate)) + |
|  | geom\_vline(data=thresholds, |
|  | mapping=aes(xintercept=threshold, color=estimate), |
|  | linetype=2) + |
|  | ggtitle("Estimated model utility as a function of threshold", |
|  | subtitle=subtitle) + |
|  | scale\_color\_manual(values=pal) + |
|  | theme(legend.position='bottom') |
|  |  |
|  | ``` |
|  |  |
|  | The best threshold is around `r format(best\_threshold, digits=2)`, which realizes an estimated utility of |
|  | \$`r dollars\_cents(total\_utility)` on the evaluation set. |
|  |  |
|  | Since utility curves estimated on raw data sets can be noisy, it might be more stable to estimate the optimal threshold on a smoothed curve, like the one we've also plotted above. In this case, the optimal thresholds estimated on the raw data and on the smoothed curve are quite close (they match to two significant figures), although the total utility estimate from the smoothed curve appears to be biased down. We'll discuss the smoothing method used here a bit later, below. |
|  |  |
|  | **## Estimating uncertainty bounds with bootstrapping** |
|  |  |
|  | Now we have an optimal threshold, which we'll call `best\_threshold`, and a estimate of the utility of that threshold. You might also want some uncertainty bars around that estimate. One way to estimate those uncertainty bars is with [bootstrap sampling](https://en.wikipedia.org/wiki/Bootstrapping\_%28statistics%29#Deriving\_confidence\_intervals\_from\_the\_bootstrap\_distribution). In fact, if we are patient, we can simulate uncertainty bars for multiple thresholds (or the entire utility curve) simultaneously. |
|  |  |
|  | Bootstrap sampling simulates having multiple evaluation sets with similar characteristics by sampling from the original data set with replacement. We generate a new data set by resampling, then call `sigr::model\_utility()` on the new data to generate a new utility curve. Repeating the procedure over and over again produces a large collection of curves. These curves give us a distribution of plausible utility values for every threshold that we are interested in -- in particular, `best\_threshold`. From this distribution, we can estimate uncertainty bounds. |
|  |  |
|  | We used the `boot::boot()` function to implement our bootstrapping. For simplicity, we use the raw percentiles from the replicants, and don't attempt to correct for subsampling bias (which should be small for large data). We've wrapped the whole procedure in a function called `estimate\_utility\_graph()`; the source for the function is on github, [here](https://github.com/WinVector/sigr/blob/main/extras/utility\_modeling/calculate\_utility\_graph.R). This function generates 1000 bootstrap estimates from the original data, and returns the relevant summary statistics. |
|  |  |
|  | An example use is as follows (we'll restart from the beginning, so the code is all in one place): |
|  |  |
|  | ```{r eval=FALSE, echo=TRUE} |
|  | library(wrapr) # misc convenience functions |
|  | library(sigr) # for model\_utility() |
|  | library(rquery) # data manipulation |
|  | library(cdata) # data manipulation |
|  | library(boot) # bootstrap sampling library |
|  | source("calculate\_utility\_graph.R") |
|  |  |
|  | # d is the data frame of model predictions and outcomes |
|  |  |
|  | # utilities |
|  | true\_positive\_value <- 100 - 5 # net revenue - cost |
|  | false\_positive\_value <- -5 # the cost of a call |
|  | true\_negative\_value <- 0.01 # a small reward for getting them right |
|  | false\_negative\_value <- -0.01 # a small penalty for having missed them |
|  | d$true\_positive\_value <- true\_positive\_value |
|  | d$false\_positive\_value <- false\_positive\_value |
|  | d$true\_negative\_value <- true\_negative\_value |
|  | d$false\_negative\_value <- false\_negative\_value |
|  |  |
|  | # estimate\_utility\_graph() defined in |
|  | # https://github.com/WinVector/sigr/blob/main/extras/utility\_modeling/calculate\_utility\_graph.R |
|  |  |
|  | unpack[plot\_thin, boot\_summary] <- estimate\_utility\_graph( |
|  | d, |
|  | prediction\_column\_name = "predicted\_probability", |
|  | outcome\_column\_name = "converted") |
|  |  |
|  | ``` |
|  |  |
|  | (You can see the full use example in [the source code for this article](https://github.com/WinVector/sigr/blob/main/extras/utility\_modeling/Utility\_Sampling\_Distribution.Rmd).) |
|  |  |
|  | The `estimate\_utility\_graph()` function returns two data frames, `boot\_summary` and `plot\_thin`. |
|  | The data frame `boot\_summary` has columns for the mean utility curve over all the bootstrap samples, as well as |
|  | curves for several key quantiles. |
|  |  |
|  | ```{r echo=TRUE} |
|  | knitr::kable(head(boot\_summary, n=3)) |
|  | ``` |
|  |  |
|  | One interesting uncertainty bound is the range between the 2.5th percentile (`q\_.0.025`) and the 97.5th percentile (`q\_0.975`), which holds 95% of the observations. With some abuse of terminology, you can consider this analogous to a "95% confidence interval" for your estimated utility. |
|  |  |
|  | Right now, the function is hard coded to return estimates at all the same thresholds used in the original utility curve. |
|  | You can use this data to get utility estimates at key threshold values, like `best\_threshold`. |
|  |  |
|  | ```{r echo=TRUE} |
|  | # find the statistics corresponding to best\_threshold |
|  | ix = which(abs(boot\_summary$threshold - best\_threshold) < 1e-5)[1] |
|  | best\_stats = boot\_summary[ix, ] |
|  | knitr::kable(best\_stats) |
|  | ``` |
|  |  |
|  | ```{r} |
|  | interval95 = with(best\_stats, c(dollars\_cents(q\_0.025), |
|  | dollars\_cents(q\_0.975))) |
|  | ``` |
|  |  |
|  | At `best\_threshold`, we estimate that 95% of the time, the total utility realized will be in the range \$`r interval95[1]` to \$`r interval95[2]`. |
|  |  |
|  | The `plot\_thin` data frame (in long form so it's easy to use with `ggplot2`) again has the mean utility curve over all |
|  | the bootstrap samples, the original utility curve from the real data, and the smoothed curve that we showed at the beginning of the article. |
|  |  |
|  | ```{r echo=TRUE} |
|  | knitr::kable(head(plot\_thin, n=3)) |
|  | unique(plot\_thin$estimate) |
|  | ``` |
|  |  |
|  | We can use these two data frames to plot the utility curve with uncertainty. Here we show the original curve, the smoothed curve, and the 50% and 95% quantile ranges around them. |
|  |  |
|  | ```{r} |
|  | # get a plot region |
|  | values <- plot\_thin[plot\_thin$estimate == 'estimated value', ] |
|  | # get optimal |
|  | best\_idx <- which.max(values$total\_value) |
|  | chosen\_threshold <- values$threshold[[best\_idx]] |
|  |  |
|  | # limit to a nice range |
|  | # threshold\_list <- values$threshold[!is.na(values$threshold)] # all |
|  | threshold\_list <- values$threshold[(!is.na(values$threshold)) & (values$total\_value >= -0.5\*max(values$total\_value))] # nice |
|  | diff\_left <- max(chosen\_threshold - threshold\_list) |
|  | threshold\_list <- threshold\_list[abs(threshold\_list - chosen\_threshold) <= 2\*diff\_left] |
|  | plot\_thin <- plot\_thin[plot\_thin$estimate != 'bootstrapped value', , drop = FALSE] |
|  |  |
|  | theoretical\_graph <- parametric\_utility\_graph( |
|  | d, |
|  | prediction\_column\_name = "predicted\_probability", |
|  | outcome\_column\_name = "converted", |
|  | true\_positive\_value = true\_positive\_value, |
|  | false\_positive\_value = false\_positive\_value, |
|  | true\_negative\_value = true\_negative\_value, |
|  | false\_negative\_value = false\_negative\_value |
|  | ) |
|  | theoretical\_graph <- theoretical\_graph[, colnames(plot\_thin)] |
|  | plot\_thin <- rbind(plot\_thin, theoretical\_graph) |
|  |  |
|  | plot\_thin <- plot\_thin[ |
|  | (complete.cases(plot\_thin)) & |
|  | (plot\_thin$threshold >= min(threshold\_list)) & |
|  | (plot\_thin$threshold <= max(threshold\_list)), ] |
|  | boot\_summary <- boot\_summary[ |
|  | (complete.cases(boot\_summary)) & |
|  | (boot\_summary$threshold >= min(threshold\_list)) & |
|  | (boot\_summary$threshold <= max(threshold\_list)), ] |
|  |  |
|  | pal <- c('#841c17', 'yellow') |
|  | ggplot() + |
|  | geom\_ribbon( |
|  | data = boot\_summary, |
|  | mapping = aes(x = threshold, ymin = q\_0.025, ymax = q\_0.975), |
|  | alpha = 0.2, |
|  | fill = '#5e8190') + |
|  | geom\_ribbon( |
|  | data = boot\_summary, |
|  | mapping = aes(x = threshold, ymin = q\_0.25, ymax = q\_0.75), |
|  | alpha = 0.3, |
|  | fill = '#263743') + |
|  | geom\_line( |
|  | data = plot\_thin, |
|  | mapping = aes(x = threshold, y = total\_value, color = estimate), |
|  | alpha = 0.8) + |
|  | scale\_color\_manual(values = pal) + |
|  | geom\_vline(xintercept = chosen\_threshold, linetype = 2, color='darkgray') + |
|  | theme\_bw() + |
|  | theme(legend.position = "none") + |
|  | ggtitle("total value as function of utility", subtitle = "(raw) bootstrapped 95% and 50% quantile ranges shown") |
|  | ``` |
|  |  |

# Conclusion

Utility is a simple and intuitive metric for selecting good classifier thresholds. In this note, we’ve shown how to estimate uncertainty bands around your utility calculations. The clarity of the original utility graph makes visualizing uncertainty quite easy. We feel this is a good tool to add to your decision-making arsenal.