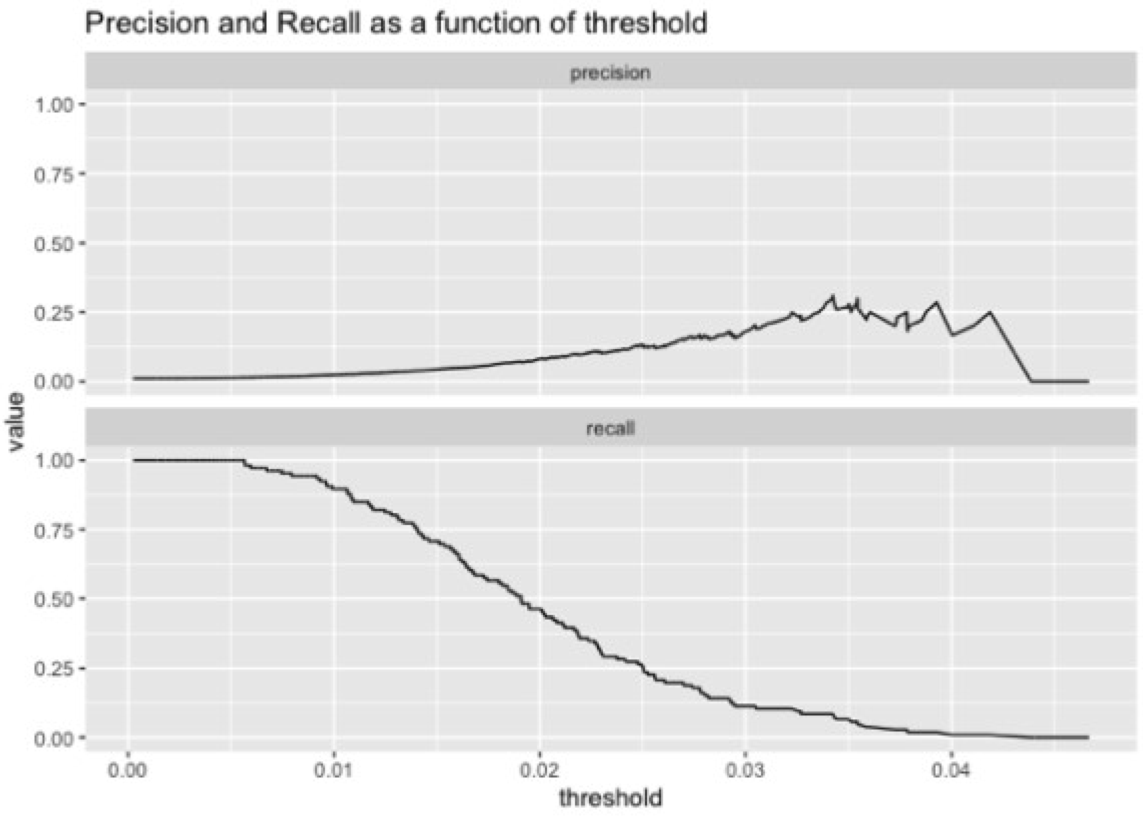
We want to contact prospects with a higher probability of converting; that is, prospects who score above a certain threshold. How how do we set that threshold?

Note that the usual default threshold of 0.5 will fail spectacularly with this model, since both positive and negative examples score quite low. This is not uncommon in low positive prevalence situations.

Picking a threshold based on model performance

We might try picking the threshold by looking at model performance. Let's look at precision (the probability that a sample scored as "true" really is true) and recall (the fraction of conversions found) as a function of threshold.

```
ThresholdPlot(d, "predicted_probability", "converted",
              title = "Precision and Recall as a function of
threshold",
              metrics = c("precision", "recall"))
```



For the sake of efficiency, we'd like to reduce the number of unsuccessful calls as much as possible, which implies we'd like a decision policy with high precision. The best precision we can get with this model occurs at a threshold of around 0.034 or 0.035 or so, and at that point the recall is quite low. Is that classifier good enough for our business needs? This can be hard to tell, as the business needs are likely in dollars, rather than in false positive/false negatives, and so on.

Picking a threshold based on model *utility*

The way to decide if a possible threshold meets business needs is to attach utilities to the decisions – right and wrong – that the model will make. To do this, we need to assign costs and rewards to contacting (or not contacting) a prospect.

Suppose every contact we make costs $5, and every successful conversion brings in $100 in net revenue. For demonstration purposes, we will also add a small penalty for every missed prospect (one penny), and a small reward for every prospect we correctly ignore (again, one penny). Let’s add those costs to our model frame. (You can leave the last two values at zero, if you prefer).

```
d$true_positive_value <- 100 - 5   # net revenue - cost
d$false_positive_value <- -5       # the cost of a call
d$true_negative_value <-  0.01     # a small reward for getting them
right
d$false_negative_value <- -0.01    # a small penalty for having missed
them
```

As we vary the decision threshold, we vary the number of prospects contacted, and the number of successful conversions. We can use the costs and rewards above to calculate the total value (or the *total utility*) realized by a decision policy over the evaluation set. The threshold that realizes the highest utility is the best threshold to use with a given model (for now, we ignore also modeling uncertainty).

The `sigr::model_utility()` function can calculate all the costs for various thresholds.

```
library(sigr)
values <- model_utility(d,
                        model_name = 'predicted_probability',
                        outcome_name = 'converted')
```

The `model_utility()` function returns a data frame with the following columns (with one example row):

```
t(values[1, ])
```

```
##                      1
## model                "predicted_probability"
## threshold            "0.0002494199"
## count_taken          "10000"
## fraction_taken       "1"
## true_positive_value  "10070"
## false_positive_value "-49470"
## true_negative_value  "0"
## false_negative_value "0"
## total_value          "-39400"
## true_negative_count  "0"
## false_negative_count "0"
## true_positive_count  "106"
## false_positive_count "9894"
```

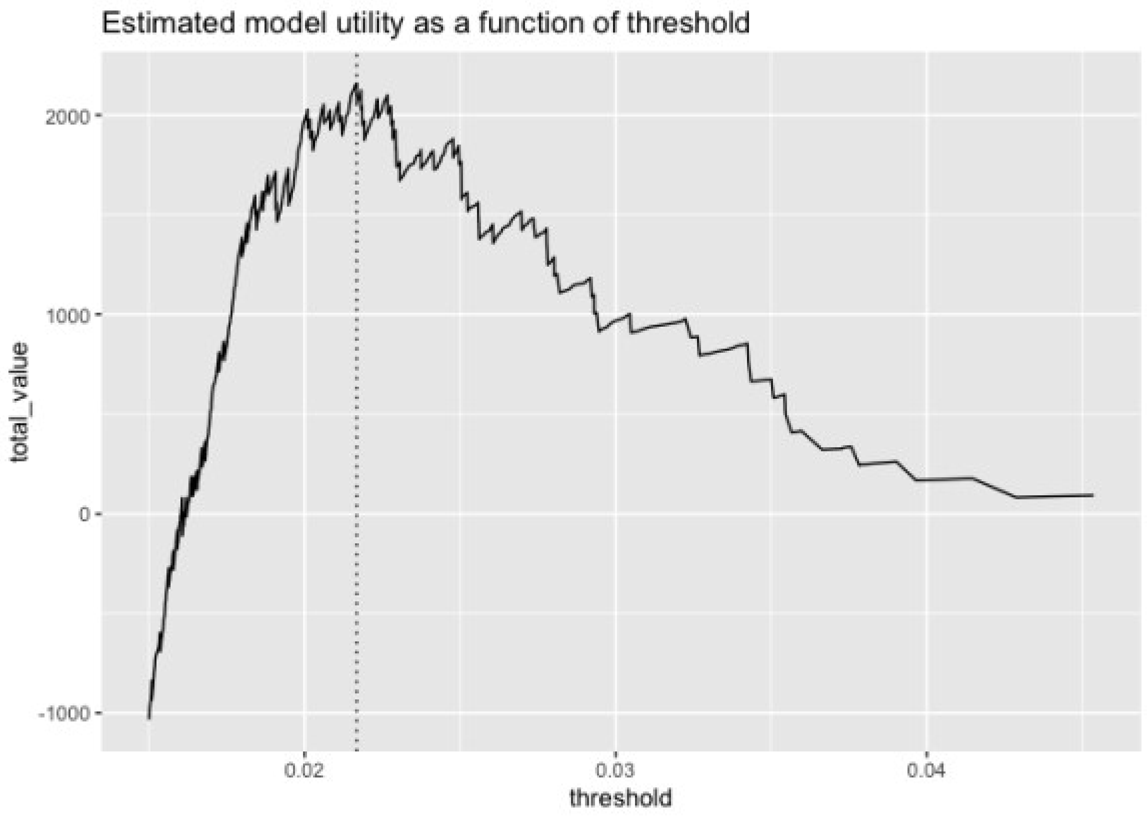
Each row of `values` returns the appropriate counts and values for a classifier rule that labels cases as TRUE when `predicted_probability >= threshold`. Now we can determine the total value returned by the model on our evaluation set, as a function of threshold.

```
library(ggplot2)
```

```
t_bound <- 0.015
vhigh <- subset(values, threshold >= t_bound) # just look at thresholds
>= t_bound
p <- ggplot(vhigh, aes(x = threshold, y = total_value)) +
  geom_line() +
  ggtitle("Estimated model utility as a function of threshold")

# find the maximum utility
max_ix <- which.max(vhigh$total_value)
best_threshold <- vhigh$threshold[max_ix]

p + geom_vline(xintercept = best_threshold, linetype=3)
```



```
# print out some information about the optimum
(best_info <- vhigh[max_ix, c("threshold", "count_taken",
                              "fraction_taken", "total_value")]) %.>%
  knitr::kable(.)
```

	threshold	count_taken	fraction_taken	total_value
9574	0.021672	427	0.0427	2159.45

The graph shows that a policy that contacts too many prospects (lower threshold) will *lose* money (negative utility), while contacting too few prospects (higher threshold) goes to zero utility. The best tradeoff with this model is a threshold of 0.022, which translates to contacting the top 4.27% of the prospects.

If we go back to the precision/recall graph, we see that this optimal threshold actually doesn't give us the policy with the highest precision, but it makes up for that by making more successful contacts. Picking the threshold from the utility calculation is easier and more reliable than just eyeballing a threshold from the abstract recall/precision graph.

Comparing to best possible performance

We can compare this model’s optimal policy to the best possible performance on this data (contacting all of the successful prospects, and only them), simply by positing a “wizard model” that scores true cases as 1.0 and false cases as 0.0. In this case any threshold between zero and one (say, 0.5) will perform the same. For this situation, the `model_utility` function will return three rows: one for the policy of contacting everyone (threshold = 0), one for threshold = 0.5, and one for the policy of contacting no one (marked as threshold = NA).

```
# add a column for the wizard model
d$wizard <- with(d, ifelse(converted, 1.0, 0.0))

# calculate the wizard's model utilities
wizard_values <- model_utility(d,
                               model_name = 'wizard',
                               outcome_name = 'converted')
wizard_values[, c("threshold", "count_taken",
                  "fraction_taken", "total_value")] %.>%
  knitr::kable(.)
```

threshold	count_taken	fraction_taken	total_value
0.0	10000	1.0000	-39400.00
0.5	106	0.0106	10168.94
NA	0	0.0000	97.88

```
# amount of potential value in this population
potential = wizard_values$total_value[2]

# what we realized
realized = best_info$total_value

# fraction realized
frac_realized = realized/potential
```

The `threshold = 0.5` row tells us that the potential value of this population sample is $10168.94, of which our model realized $2159.45, or 21.2% of the total available value.

Advanced: Variable Utilities

In the above example, we assigned constant utilities and costs to the data set. Here, instead of assuming a fixed revenue of $100 dollars on conversions, we’ll assume that projected potential revenue varies by prospect (with an average of $100). This potential value could be contract sizes that we are bidding on, which varies from prospect to prospect.

```
# replace the true positive value with a varying value
d$true_positive_value <- rnorm(nrow(d), mean = 100, sd = 25) - 5   #
revenue - cost

# recalculate the values and potential
values <- model_utility(d,
                        model_name = 'predicted_probability',
                        outcome_name = 'converted')
```

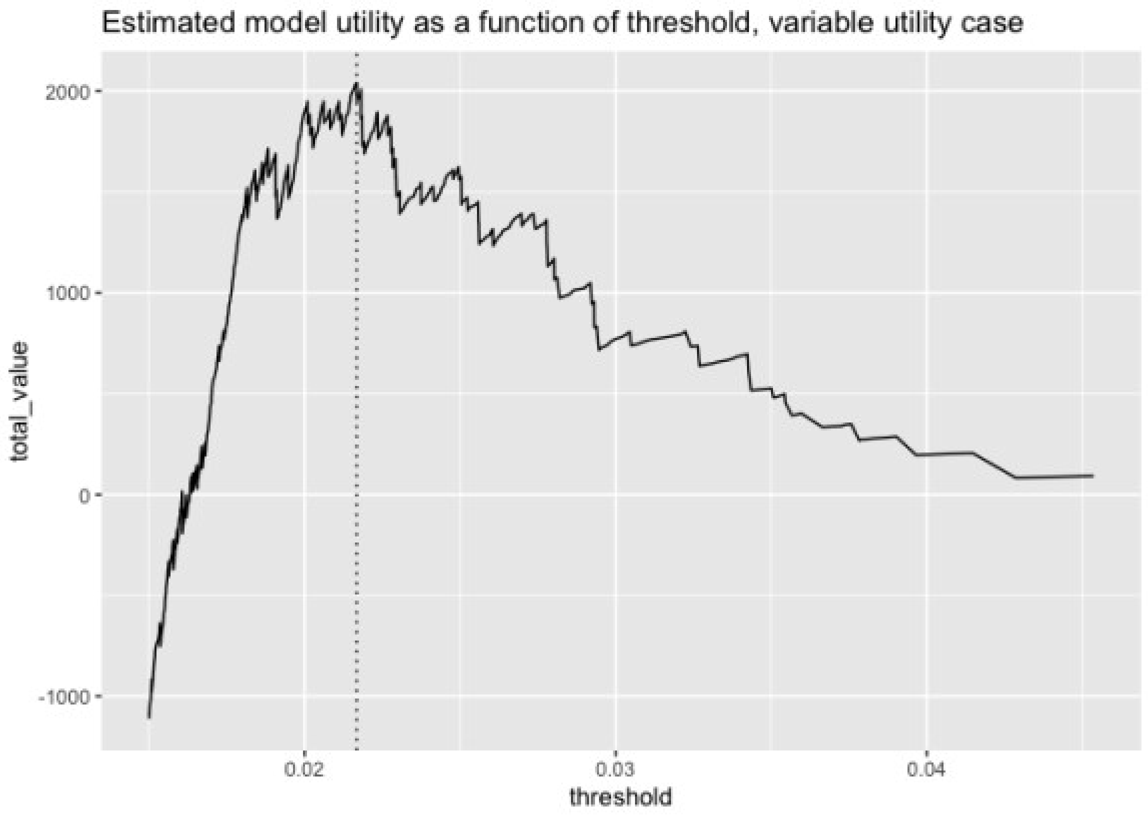
```
wizard_values <- model_utility(d,
                               model_name = 'wizard',
                               outcome_name = 'converted')
```

Here we replot the utility curve.

```
vhigh <- subset(values, threshold >= t_bound) # just look at thresholds
>= t_bound
p <- ggplot(vhigh, aes(x = threshold, y = total_value)) +
  geom_line() +
  ggtitle("Estimated model utility as a function of threshold, variable
utility case")

# find the maximum utility
max_ix <- which.max(vhigh$total_value)
best_threshold <- vhigh$threshold[max_ix]

p + geom_vline(xintercept = best_threshold, linetype=3)
```



```
# print out some information about the optimum
vhigh[max_ix, c("threshold", "count_taken", "fraction_taken",
"total_value")] %.>%
  knitr::kable(.)
```

	threshold	count_taken	fraction_taken	total_value
9574	0.021672	427	0.0427	2041.251

```
# compare to potential
wizard_values[2, c("threshold", "count_taken", "fraction_taken",
"total_value")] %.>%
  knitr::kable(.)
```

	threshold	count_taken	fraction_taken	total_value
2	0.5	106	0.0106	10136.36

Reporting Uncertainty

Up until now, we’ve returned point utility estimates, based on an evaluation set. We assume that this evaluation set is representative of the general population that we will be applying this model to, but of course there will be some variation. For this reason, we’d like to have an idea of the uncertainty of our utility estimates. In our next note, we will deal with estimating and reporting uncertainty.

Conclusion

Thinking in terms of utility is simple, because it’s already the language of your business partners. Utility estimates help you choose the best and most useful classifier rules for your probability models. Here, we’ve shown how to select a classifier threshold directly from policy utility, using `sigr::model_utility()`.