Pilot 3

Alex Kale

8/8/2019

Recap of Pilot 2

In pilot 2 we asked participants to view distributions of predicted performance for two widget manufacturing machines and judge the probability that a new machine would produce more (defective) widgets the old machine (a.k.a., probability of superiority). We then asked them whether or not they would buy the new machine, where the utility optimal decision rule was such that they should buy the new machine on half of trials where effect sizes are most extreme. Each participant comlpeted 24 trials at different levels of effect size and using one of three visualizations: means and intervals, means only, or HOPs. Where the ground truth probability of superiority was above 50%, the task was framed as a possibility of gaining a contract above some threshold number of widgets produced. Where the ground truth probability of superiority was below 50%, the task was framed as a possibility of losing a contract above some threshold number of defective widgets produced.

There were a handful of issues with this study design: 1. The elicitation of probability of superiority judgments was confusing. Participants often responded on the wrong side of 50%, yielding poor data quality and inferential validity. 2. Our strategy for sampling ground truth effect size was optimized for modeling probability of superiority judgments but not for modeling decision-making. The values we sampled did not cluster near the utility optimal decision threshold, yielding psychometric function fits with higher entropy than is ideal. 3. Manipulating the baseline probability of gaining or keeping the contract with the old machine proved fruitless. Model comparisons indicate that including this manipulation as a predictor was not worth the increased risk of overfitting.

To remedy these issues, we redesign the interface to reduce confusion. We change the probability of superiority elicitation from a scale of 0 to 100 to a scale of 50 to 100. We also change the task from machine purchasing decisions to a fantasy sports scenario to make it more engaging. Last, we remove the baseline manipulation to simplify the study design.

Pilot 3

In pilot three, we the structure of the incentivized decision task we had for pilot two, but we change the narrative around the task.

Betting Task: Scenario and Payoff Scheme

In our new task, users play a fantasy sports game where they win or lose awards depending on how many points their team scores or gives up, respectively. They are presented with charts comparing how many points their team is predicted to score or give up with or without adding a new player to their team.

In the gain framing trails, users are told they will *win an award* worth \$X if their team *scores more than 100 points* (an arbitrary number). They can improve their chances of getting that award if they pay \$C for a new player The optimal decision rule in this case maximizes expected gains.

$$X * p(award| \sim player) < X * p(award|player) - C$$

If we assume a constant ratio between the value of the award and the cost of the new player $K = \frac{X}{C}$, the decision rule can be expressed as in terms of the difference in probability of winning the award with and without a new player.

$$p(award|player) - p(award| \sim player) > \frac{1}{K}$$

The more the new player increases their chances of winning the award, the more evidence there is in favor of intervening by paying for the new player.

The loss framing trails are similarly set up. Users are told they will *lose an award* worth \$X if their team *gives up more than 75 points* (an arbitrary number). They can improve their chances of keeping that award if they pay \$C for a new player. The optimal decision rule in this case minimizes expected losses.

$$X * p(\sim award| \sim player) > X * p(\sim award|player) + C$$

Again, if we assume a constant ratio between the value of the award and the cost of the new player $K = \frac{X}{C}$, the decision rule can be expressed as in terms of the difference in probability of losing the award with and without a new player

$$p(\sim award| \sim player) - p(\sim award|player) > \frac{1}{K}$$

The more the new player decrease their chances of losing the award, the more evidence there is in favor of intervening by paying for the new player.

Each trial, users will receive feedback based on their decision and a simulated outcome regarding the award in question. We will tally the dollar value of awards won/kept minus the cost of new players across trials to determine the user's fantasy sports account balance. Users will receive a bonus through MTurk that is porportional to the value of their account at the end of the HIT.

Visualization Conditions

Users will be shown predicted distributions of how many points their team will score or give up with vs without the new player. Number of points will be visualized as *means only, means with intervals, HOPs with means, densities, quantile dotplots, intervals only, and HOPs*. These conditions span a continuum of how much the make the mean available to users, from emphasizing the mean to only encoding it implicitly.

Data Conditions

We manipulate the probability of the team scoring or giving up more points with vs without the new player (p_superiority). We employ two sampling strategies, one which optimizes for each of the two questions we ask participants: 1. Linear intervals in logodds units to give perceptually uniform steps in probability of superiority. 2. Probability of superiority values near the utility optimal decision threshold (i.e., p_superiority == [0.13, 0.87]).

When p_superiority is greater than 0.5, the decision task is framed as a gain scenario where the user's team needs to score at least 100 points to win an award. When p_superiority is less than 0.5, the decision task is framed as a loss scenario where the user's team needs to give up fewer than 75 points to keep an award.

```
# linear sampling of log odds for full span of ground truth probability of superiority between 0.025 a
nd 0.975
n_trials.full_span <- 20
logodds.full_span <- seq(log(0.025 / (1 - 0.025)), log(0.975 / (1 - 0.975)), length.out = n_trials.ful
l_span)

# linear sampling of log odds near the decision threshold (p_superiority == [0.13, 0.87])
n_trials.near_threshold <- 8
logodds.near_threshold <- c(seq(log(0.1 / (1 - 0.1)), log(0.2 /(1 - 0.2)), length.out = n_trials.near_
threshold / 2), # near threshold for loss frame
                            seq(log(0.8 / (1 - 0.8)), log(0.9 /(1 - 0.9)), length.out = n_trials.near_
threshold / 2)) # near threshold for gain frame

# combine the sampling strategies and convert from log odds to probability of superiority
logodds <- sort(c(logodds.full_span, logodds.near_threshold))
p_superiority <- 1 / (1 + exp(-logodds))
n_trials <- length(p_superiority)

print(p_superiority)
```

```
##  [1] 0.02500000 0.03633635 0.05253624 0.07539348 0.10000000 0.10707153
##  [7] 0.12709249 0.14990182 0.16021834 0.20000000 0.20591399 0.27605902
## [13] 0.35928769 0.45194404 0.54805596 0.64071231 0.72394098 0.79408601
## [19] 0.80000000 0.83978166 0.85009818 0.87290751 0.89292847 0.90000000
## [25] 0.92460652 0.94746376 0.96366365 0.97500000
```

This time around, we set the baseline probability of winning/keeping the award without the new player to a constant value of 0.5. The team is as likely as a coin flip to win or keep the award without the new player. This represents the scenario where there is the maximum uncertainty about outcomes without intervention.

```
# baseline probability of winning/keeping an award contract without the new player
baseline <- c(.5) # previously c(.15, .5, 8.5)

# initialize data conditions dataframe
conds_df <- data.frame(
  "p_superiority" = rep(p_superiority, length(baseline)),
  "baseline" = sort(rep(baseline, length(p_superiority))))

head(conds_df)
```

```
##    p_superiority baseline
## 1    0.02500000       0.5
## 2    0.03633635       0.5
## 3    0.05253624       0.5
## 4    0.07539348       0.5
## 5    0.10000000       0.5
## 6    0.10707153       0.5
```

As stated above, we set the threshold for winning the award in the gain frame at 100 points and the threshold for keeping the award in the loss frame at 75 points.

```
# label gain vs loss framing trials based on p_superiority, and add award thresholds
conds_df <- conds_df %>%
  mutate(frame = if_else(p_superiority > .5, "gain", "loss"),
         threshold = if_else(frame == "gain",
                             100, # points scored
                             75)) # points given up

head(conds_df)
```

```
##    p_superiority baseline frame threshold
## 1    0.02500000       0.5  loss        75
## 2    0.03633635       0.5  loss        75
## 3    0.05253624       0.5  loss        75
## 4    0.07539348       0.5  loss        75
## 5    0.10000000       0.5  loss        75
## 6    0.10707153       0.5  loss        75
```

We control the standard deviation of the distribution of the difference in points between the team with and without the new player (sd_diff) by setting it to 15. In the gain framing this is 15 points scored. In the loss framing, this is 15 points given up. We can think of this variable as constant across trials. We then derive the mean difference in the number of points scored by the team with minus without the new player (mean_diff) from sd_diff and p_superiority.

```
# add columns for the mean and standard deviation of the difference in the number of points for the te
am with vs without the new player
# depending on the gain vs loss frame, these values represent points scored vs points given up
conds_df <- conds_df %>%
  mutate(sd_diff = 15, # std(new - old)
         mean_diff = sd_diff * qnorm(p_superiority)) # mean(new - old)

head(conds_df)
```

```
##    p_superiority baseline frame threshold sd_diff mean_diff
## 1    0.02500000       0.5  loss        75      15 -29.39946
## 2    0.03633635       0.5  loss        75      15 -26.92321
## 3    0.05253624       0.5  loss        75      15 -24.31117
## 4    0.07539348       0.5  loss        75      15 -21.55136
## 5    0.10000000       0.5  loss        75      15 -19.22327
## 6    0.10707153       0.5  loss        75      15 -18.63380
```

Now we calculate the summary statistics for the team with and without the new player, making the dataframe double its length up to this point. We derive the standard deviation of the points scored by the teams with and without the new player (sd) from sd_diff, variance sum law, and the assumption that the teams with or without the new player have equal and independent variances. We derive the mean number points scored by the teams with and without the new player (mean) from the threshold for winning/keeping the award, the sd of points for each version of the team, and the mean_diff between the number of points for with minus without the new player. We derive the probability of winning/keeping the award from the threshold, mean, and sd.

```
# double the length of the dataframe to add information per version of the team, creating a stimulus d
ataframe with a row per distribution to visualize
stim_df <- map_df(seq_len(2), ~conds_df)
stim_df$team <- as.factor(sort(rep(c("with_player","without_player"), length(stim_df$p_superiority) /
2)))


# add columns for the mean and standard deviation of points for each team and the probability of winni
ng/keeping the award
stim_df <- stim_df %>%
  mutate(sd = sqrt(stim_df$sd_diff ^ 2 / 2), # assume equal and independent variances
        mean = if_else(team == "without_player",
                       if_else(frame == "gain", # team without the new player is at baseline
                              threshold - sd * qnorm(1 - baseline),
                              threshold - sd * qnorm(baseline)),
                      if_else(frame == "gain", # team with new player is at difference from baseline
                              threshold - sd * qnorm(1 - baseline) + mean_diff,
                              threshold - sd * qnorm(baseline) + mean_diff)),
        p_award = if_else(frame=="gain", # probability of exceeding threshold to win/keep award
                         1 - pnorm((threshold - mean)/sd),
                         pnorm((threshold - mean)/sd)))

# spread values per machine across columns to get back to a conditions dataframe one row per trial
conds_df <- stim_df %>% # explanation: https://kieranhealy.org/blog/archives/2018/11/06/spreading-mult
iple-values/
  gather(variable, value, -(p_superiority:team)) %>%
  unite(temp, team, variable) %>%
  spread(temp, value)

head(conds_df)
```

```
##   p_superiority baseline frame threshold sd_diff mean_diff
## 1    0.02500000      0.5  loss        75      15 -29.39946
## 2    0.03633635      0.5  loss        75      15 -26.92321
## 3    0.05253624      0.5  loss        75      15 -24.31117
## 4    0.07539348      0.5  loss        75      15 -21.55136
## 5    0.10000000      0.5  loss        75      15 -19.22327
## 6    0.10707153      0.5  loss        75      15 -18.63380
##   with_player_mean with_player_p_award with_player_sd without_player_mean
## 1         45.60054           0.9972127        10.6066                  75
## 2         48.07679           0.9944311        10.6066                  75
## 3         50.68883           0.9890494        10.6066                  75
## 4         53.44864           0.9789172        10.6066                  75
## 5         55.77673           0.9650368        10.6066                  75
## 6         56.36620           0.9605250        10.6066                  75
##   without_player_p_award without_player_sd
## 1                    0.5           10.6066
## 2                    0.5           10.6066
## 3                    0.5           10.6066
## 4                    0.5           10.6066
## 5                    0.5           10.6066
## 6                    0.5           10.6066
```

This results in an experimental design where the probability of winning/keeping the award with the new player increases monotonically with p_superiority. This means that users should intervene only at extreme values of p_superiority. Even though the decision rule is not defined in terms of p_superiority, users can user effect size as a proxy for the decision task.
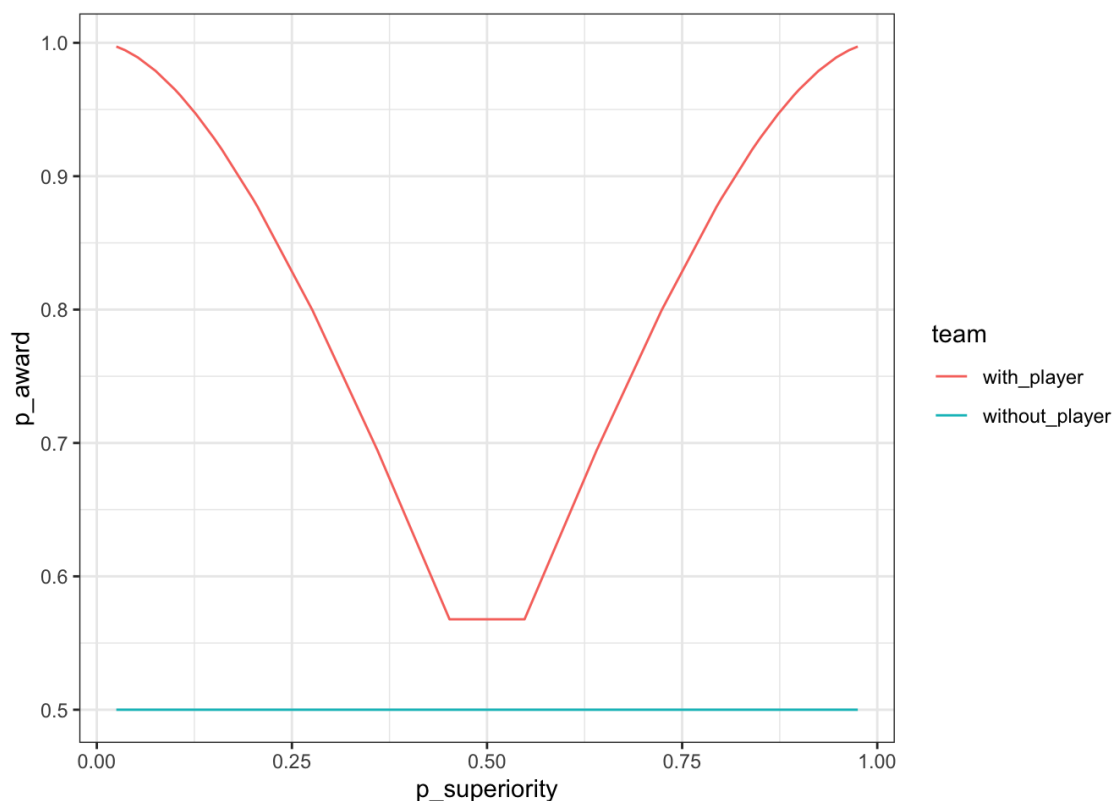
Contract Values and Thresholds

Since we hold the cost of the new player constant at \$1M, we can think of K as the *value of the award in millions of dollars*. We need to set K so that there is an *equal number of trials where users should vs shouldn't intervene*. A value that guarantees this balance for our sample of probability of superiority values is 2.25.

```
# ratio (K) of value of contract (X) over cost of intervention (C)
conds_df <- conds_df %>% mutate(K = 2.25)
```

Let's check that we have an equal number of trials where intervening is and is not the optimal choice. We want to make sure that this balance is maintained across both levels framing.

```
# determine whether or not intervention is utility optimal on each trial
conds_df <- conds_df %>%
  mutate(should_intervene = if_else(frame == "gain",
                                    (with_player_p_award - without_player_p_award) > (1 / K), # gain f
raming decision rule
                                    ((1 - without_player_p_award) - (1 - with_player_p_award)) > (1 /
 K)) # loss framing decision rule
  )

conds_df %>%
  group_by(frame) %>%
  summarise(intervene = sum(should_intervene), n_trials = n())
```

```
## # A tibble: 2 x 3
##   frame intervene n_trials
##   <chr>     <int>    <int>
## 1 gain          7       14
## 2 loss          7       14
```

What exactly are the values of probability of superiority at these thresholds?

```
conds_df <- conds_df %>%
  mutate(
    p_sup_threshold = if_else(frame == "gain",
                              pnorm(sqrt(sd_diff ^ 2 / 2) / sd_diff * (qnorm((1 / K) + baseline) + qno
rm(baseline))),
                              pnorm(-sqrt(sd_diff ^ 2 / 2) / sd_diff * (qnorm((1 / K) + baseline) - qn
orm(baseline))))
  )

# unique(conds_df$p_sup_threshold)
conds_df %>%
  group_by(frame) %>%
  summarise(threshold = unique(p_sup_threshold))
```

```
## # A tibble: 2 x 2
##   frame threshold
##   <chr>     <dbl>
## 1 gain      0.870
## 2 loss      0.130
```

Expected Bonuses

We also need to set the *starting value of the fantasy sports account* and the *exchange rate* of actual dollars in MTurk bonus per million of dollars in account value.

First, we *set the starting value of the user's account equal to the maximum possible amount participants could lose*, if they buy the new player every trial and always fail to win/keep the award

```
# starting value depends on maximum possible loss (if they pay for the new machine every time and neve
r gain\keep the contract)
conds_df <- conds_df %>% mutate(starting_account_value = n_trials + n_trials / 2 * K)

conds_df %>% group_by(frame, K, starting_account_value) %>% summarise()
```

```
## # A tibble: 2 x 3
## # Groups:   frame, K [2]
##   frame     K starting_account_value
##   <chr> <dbl>                  <dbl>
## 1 gain   2.25                   59.5
## 2 loss   2.25                   59.5
```

Next, *we set the exchange rate to mainain a consistent expected bonus of $3 if participants make the optimal decision on every trial*.

```r
# add expected change in account value on each trial to conditions dataframe
conds_df <- conds_df %>%
  mutate(expected_payoff = if_else(frame == "gain",
                                   # gain frame
                                   if_else(should_intervene,
                                           with_player_p_award * K - 1,
                                           without_player_p_award * K),
                                   # loss frame
                                   if_else(should_intervene,
                                           (1 - with_player_p_award) * (-1 * K) - 1,
                                           (1 - without_player_p_award) * (-1 * K))))

# set exchange rates based on the expected account value at the end of the experiment
payoff_df <- conds_df %>% group_by(K, starting_account_value) %>%
  summarise(
    # expected and max account value
    expected_account_value = mean(starting_account_value) + sum(expected_payoff),
    max_account_value = mean(starting_account_value) + n_trials / 2 * mean(K), # if they always win/ke
ep the award and never buy the new player
    # bonus stats with exhange rate based on constant expected bonus of $3
    exchange_rate = round(3 / expected_account_value, 5), # five decimal places are necessary to avoid
substantial rounding error
    expected_bonus = expected_account_value * exchange_rate,
    max_bonus = max_account_value * exchange_rate)

# merge these payoff variables into the conditions dataframe
conds_df <- conds_df %>% full_join(payoff_df, by = c("K","starting_account_value"))

knitr::kable(payoff_df)
```

K	starting_account_value	expected_account_value	max_account_value	exchange_rate	expected_bonus	max_bonus
2.25	59.5	60.49288	91	0.04959	2.999842	4.51269