Pilot 4

Alex Kale

8/8/2019

Pilot 4

In pilot four, we drop the framing manipulation so that the objective of the decision task is always to maximize gains.

Betting Task: Scenario and Payoff Scheme

Users play a fantasy sports game where they win awards depending on how many points their team scores. They are presented with charts comparing how many points their team is predicted to score with or without adding a new player to their team.

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 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.

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 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 with vs without the new player. Uncertainty about the predicted number of points scored will be visualized as *densities, quantile dotplots, intervals, HOPs*. Each user will complete a block of trials with the uncertainty encoding alone and then a second set of trials with extrinsic marks added to encode the mean. 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 == 0.87).

We only sample p_superiority values greater than 0.5, where the new player is expected to *improve* the team's preformance. 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. In previous pilots, we sample values of p_superiority below 0.5 and framed the decision task as loss aversion. However, we remove these trials to make the task more straightforward.

```
# linear sampling of log odds for full span of ground truth probability of superiority b
etween 0.525 and 0.975
n_trials.full_span <- 10
logodds.full_span <- seq(log(0.525 / (1 - 0.525)), log(0.975 / (1 - 0.975)), length.out
 = n_trials.full_span)

# linear sampling of log odds near the decision threshold (p_superiority == 0.87])
n_trials.near_threshold <- 4
logodds.near_threshold <- seq(log(0.8 / (1 - 0.8)), log(0.9 /(1 - 0.9)), length.out = n_
trials.near_threshold)

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

print(p_superiority)
```

```
##  [1] 0.5250000 0.6215249 0.7092960 0.7837931 0.8000000 0.8397817 0.8434139
##  [8] 0.8729075 0.8889237 0.9000000 0.9224232 0.9464285 0.9633011 0.9750000
```

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 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))),
  "threshold" = 100)

head(conds_df)
```

```
##   p_superiority baseline threshold
## 1     0.5250000      0.5       100
## 2     0.6215249      0.5       100
## 3     0.7092960      0.5       100
## 4     0.7837931      0.5       100
## 5     0.8000000      0.5       100
## 6     0.8397817      0.5       100
```

We also want to create stimuli for the practice trials. To make these trials easy, we choose probability of superiority values near 1. This way it should be obvious that the new player is worth the cost.

```
# create df containting rows for practice trials
prac_df <- data.frame(
  "p_superiority" = c(.999),
  "baseline" = c(.5),
  "threshold" = c(100))
# append to conditions dataframe
conds_df <- rbind(conds_df, prac_df)

head(prac_df)
```

```
##   p_superiority baseline threshold
## 1         0.999      0.5       100
```

Since judging probability of superiority might be difficult for participants, we are including a mock task to help them understand what we are asking. We ask them judge a case where probability of superiority is 50%.

```
# create df containting rows for mock trial in each condition
mock_df <- data_grid(conds_df, p_superiority = c(.5), baseline = c(.5), threshold = c(10
0))

# add to conds_df
conds_df <- rbind(conds_df, mock_df)

print(mock_df)
```

```
## # A tibble: 1 x 3
##   p_superiority baseline threshold
##           <dbl>    <dbl>     <dbl>
## 1           0.5      0.5       100
```

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 poi
nts for the team with vs without the new player
conds_df <- conds_df %>%
  mutate(sd_diff = 15, # std(with - without)
         mean_diff = sd_diff * qnorm(p_superiority)) # mean(with - without)

head(conds_df)
```

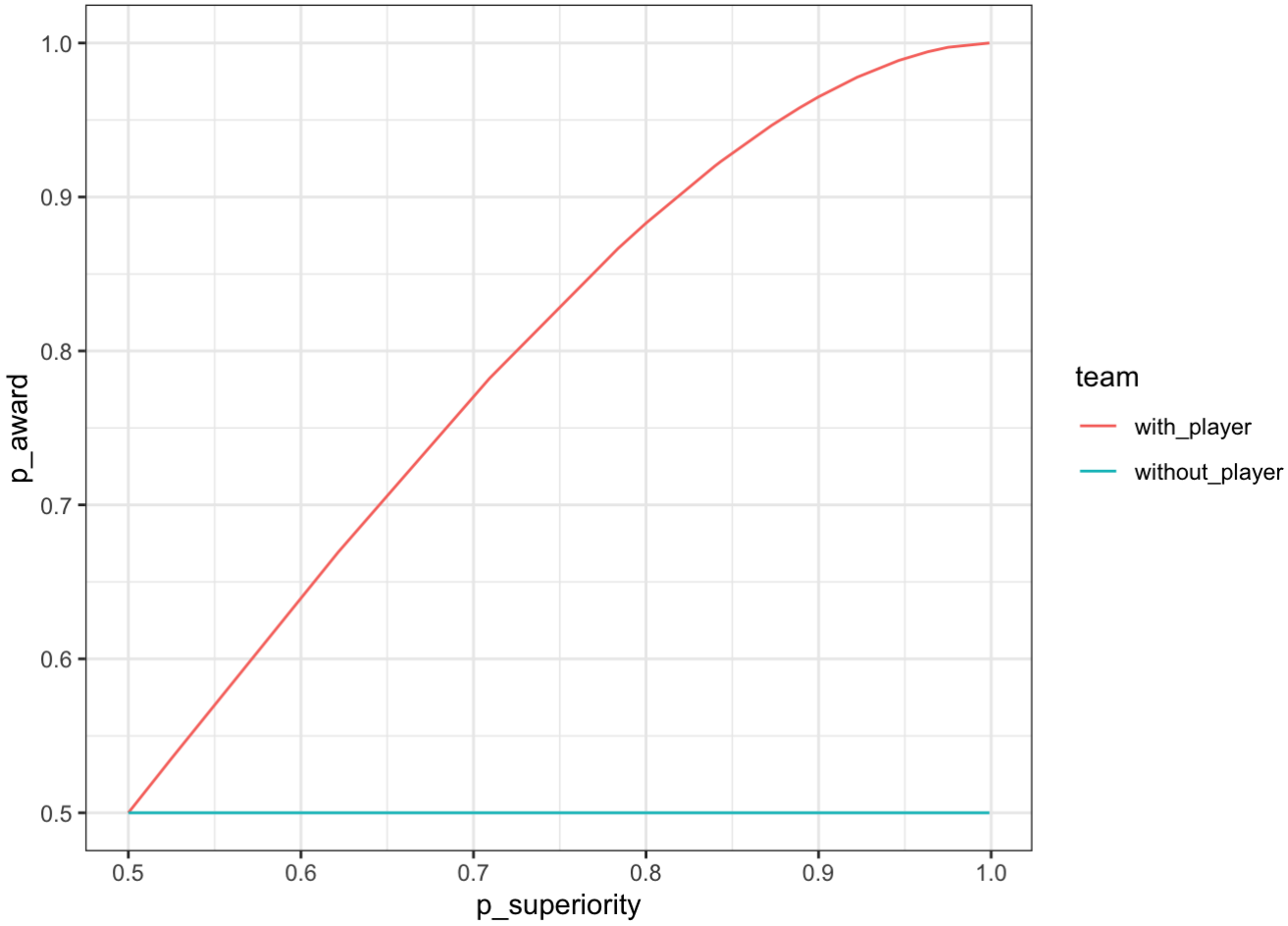
```
##   p_superiority baseline threshold sd_diff  mean_diff
## 1     0.5250000      0.5       100      15  0.9406017
## 2     0.6215249      0.5       100      15  4.6423208
## 3     0.7092960      0.5       100      15  8.2699398
## 4     0.7837931      0.5       100      15 11.7760197
## 5     0.8000000      0.5       100      15 12.6243185
## 6     0.8397817      0.5       100      15 14.9034139
```

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 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 the award from the threshold, mean, and sd.

```
# double the length of the dataframe to add information per version of the team, creatin
g a stimulus dataframe 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_s
uperiority) / 2)))

# add columns for the mean and standard deviation of points for each team and the probab
ility of winning 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",
                        # team without the new player is at baseline
                        threshold - sd * qnorm(1 - baseline),
                        # team with new player is at difference from baseline
                        threshold - sd * qnorm(1 - baseline) + mean_diff),
         p_award = pnorm((mean - threshold) / sd) # probability of exceeding threshold to
win award
  )

# 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-multiple-values/
  gather(variable, value, -(p_superiority:team)) %>%
  unite(temp, team, variable) %>%
  spread(temp, value)

head(conds_df)
```

```
##   p_superiority baseline threshold sd_diff  mean_diff with_player_mean
## 1     0.5000000      0.5       100      15  0.0000000         100.0000
## 2     0.5250000      0.5       100      15  0.9406017         100.9406
## 3     0.6215249      0.5       100      15  4.6423208         104.6423
## 4     0.7092960      0.5       100      15  8.2699398         108.2699
## 5     0.7837931      0.5       100      15 11.7760197         111.7760
## 6     0.8000000      0.5       100      15 12.6243185         112.6243
##   with_player_p_award with_player_sd without_player_mean
## 1           0.5000000        10.6066                 100
## 2           0.5353322        10.6066                 100
## 3           0.6691917        10.6066                 100
## 4           0.7822155        10.6066                 100
## 5           0.8665552        10.6066                 100
## 6           0.8830224        10.6066                 100
##   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 the award with the new player increases monotonically with p_superiority. This means that users should intervene only at high 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. This includes the mock and practice trial stimuli at probability of superiority values equal to 0.5 and 0.999, respectively. We will place one of these two at random in the middle of each block of the study as an attention check.

```
# determine whether or not intervention is utility optimal on each trial
conds_df <- conds_df %>%
  mutate(should_intervene = (with_player_p_award - without_player_p_award) > (1 / K)) #
 decision rule

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

```
##   intervene n_trials
## 1         8       16
```

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

```
conds_df <- conds_df %>%
  mutate(p_sup_threshold = pnorm(sqrt(sd_diff ^ 2 / 2) / sd_diff * (qnorm((1 / K) + base
line) + qnorm(baseline))))

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

```
##   threshold
## 1 0.8700391
```

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 the award.

```
# starting value depends on maximum possible loss (if they pay for the new machine every
time and never gain\keep the contract)
conds_df <- conds_df %>% mutate(starting_account_value = (2 * n_trials + 2) * K)

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

```
## # A tibble: 1 x 2
## # Groups:   K [1]
##       K starting_account_value
##   <dbl>                  <dbl>
## 1  2.25                   67.5
```

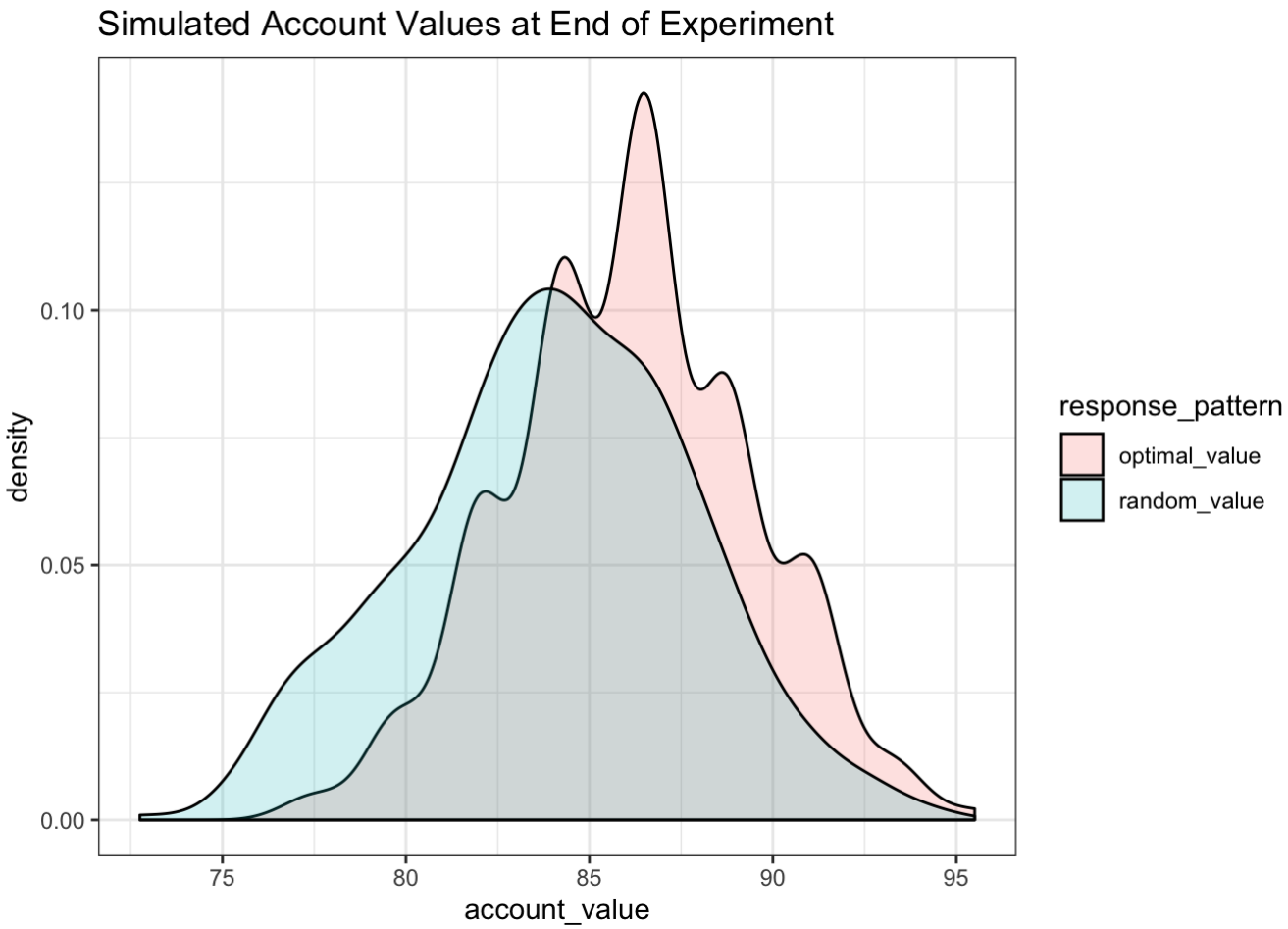
Next, *we set the exchange rate to range from $0 to $4 depending on decision quality*. To figure out how to calculate bonuses, we want to know what account values would look like at the end of the experiment if users guessed on every trial vs if they made the optimal choice on every trial. To learn this, we'll run a simulation of two response patterns: random guessing vs utility optimal decision-making.

```
# set up for simulation
n_iter <- 500
simulation_df <- NULL

# generate outcome, whether award is won/kept or not
outcome <- function(intervene, p_with, p_without) {
  if (intervene) {
    return(runif(1) <= p_with)
  } else {
    return(runif(1) <= p_without)
  }
}

for (i in 1:n_iter) {
  temp <- conds_df %>%
    mutate(
      # generate payoff for random guess
      random_guess = as.logical(rbinom(n(), 1, 0.5)),
      random_outcome = as.logical(pmap(list(random_guess, with_player_p_award, without_p
layer_p_award), outcome)),
      random_payoff = if_else(random_outcome,
                              K - random_guess,
                              as.numeric(-random_guess)),
      random_correct = (random_guess == should_intervene),
      # generate payoff for optimal guess
      optimal_outcome = as.logical(pmap(list(should_intervene, with_player_p_award, with
out_player_p_award), outcome)),
      optimal_payoff = if_else(optimal_outcome,
                               K - should_intervene,
                               as.numeric(-should_intervene)),
      optimal_correct = TRUE
    ) %>%
    summarise(
      iter = i,
      random_value = mean(starting_account_value) + sum(random_payoff),
      random_correct = sum(random_correct),
      optimal_value = mean(starting_account_value) + sum(optimal_payoff),
      optimal_correct = sum(optimal_correct)
    )
  simulation_df = rbind(simulation_df, temp)
}

head(simulation_df)
```

```
##   iter random_value random_correct optimal_value optimal_correct
## 1    1        82.00              8         84.25              16
## 2    2        83.50             11         86.50              16
## 3    3        80.25              6         88.75              16
## 4    4        76.50              7         82.00              16
## 5    5        83.50              7         84.25              16
## 6    6        81.75             10         82.00              16
```

Let's plot the results of our simulation.

```
simulation_df %>%
  select(iter, random_value, optimal_value) %>%
  gather(response_pattern, account_value, -iter) %>%
  ggplot(aes(x = account_value, fill = response_pattern)) +
  geom_density(alpha = 0.2) +
  theme_bw() +
  labs(
    title = "Simulated Account Values at End of Experiment"
  )
```



We can see that account values aren't reliably different between these two strategies. This is problem for differentiating between good and poor performance.

In order to make the incentives more fair, we'll try setting the low end of likely account values under the optimal response distribution (i.e., an account value of 80) to correspond to no bonus. Above that every unit of company value should count for $0.25 with a maximum bonus of $4. This bonus structure will reward performance the most for account values that are unlikely to be obtainted by random guessing. However, luck still has a large impact on this payoff scheme.

```
# set exchange rate and cutoff
cutoff <- 80
exchange <- 0.25

simulation_df %>%
  select(iter, random_value, optimal_value) %>%
  gather(response_pattern, account_value, -iter) %>%
  mutate(bonus = (account_value - cutoff) * exchange) %>%
  ggplot(aes(x = bonus, fill = response_pattern)) +
  geom_density(alpha = 0.2) +
  theme_bw() +
  labs(
    title = "Bonuses for Simulated Performance"
  )
```