Supplementary Information for

Social Endorsement Cues and Political Participation

Matching to Voting Records

To choose which states to validate, we identified those states that provided (for research purposes) first names, last names, and full birthdates in publicly available voting records. From these, we chose a set that minimized cost per population, but allowed us to detect a 0.5% effect with 80% power given a treatment rate of 98% and a turnout rate of 40% based on rough estimates. The cost of state records varied from $0 to $1500 per state. We excluded records from Texas because they had systematically excluded some individuals from their voting records (specifically, they did not report on the voting behavior of people that had abstained in the four prior elections). The resulting list of states included Arkansas, California, Connecticut, Florida, Kansas, Kentucky, Missouri, Nevada, New Jersey, New York, Oklahoma, Pennsylvania, and Rhode Island. These states account for about 40% of all registered voters in the U.S., and their 4 records yielded 6,338,882 matched observations of voters and abstainers that we could use to compare to treatment categories from the experiment.

About 1 in 3 users were successfully matched to voter records (success depends on many factors, including voting eligibility, rates of registration, and so on). It is important to note that the match rate for our study is lower than the match rates in many other GOTV studies, in which more than 50% of users are matched. The primary reason for the low match rate is the age distribution of Facebook users; because the population of Facebook users shows positive skew relative to the country in general (i.e., Facebook users are younger), and young people are less likely to be registered voters, we were able to match fewer records. Additionally, as in other studies in which individuals self-enter data, matches are more difficult due to a lack of consistency in name conventions in the voter file and Facebook (for instance, a voter may be listed as “Lucille” in the voter record and “Lucy” in Facebook). All information was discarded after we finished the data analysis.

In order to match information in Facebook to public voting records, we relied on a group-level matching procedure that preserves the privacy of individual actions while still allowing statistical analysis to be conducted at the individual level. We matched users to individuals on the registration list in the same state by first name, last name, and date of birth (dropping all instances that had duplicates) and set the level of error in individual assignments to be 5%. This means that a matched user identified as a voter had a 5% chance of being classified as an abstainer, and vice versa. For all estimates that use validated voting we correct the standard errors to account for the 5% error rate.

We give an example of the R code for the group-level matching procedure at the end of the Supplementary file.

Distribution of Key Variables and Balance Testing

Table S1 shows summary statistics for age, sex, and the following variables:

- Identity as a Partisan. Respondents can choose to identify their partisanship. Particular party variables (Democrat and Republican) were coded as a 1 when the name of the party appeared in the user's political views and 0 otherwise.

- Ideology. Facebook users can write in their political ideology in an open-ended response box. Particular ideology variables (Liberal and Conservative) were coded as a 1 when the ideological label appeared in the user's political views and 0 otherwise.

- Expressed Voting. For those respondents in the two treatment conditions, the site recorded when the respondent clicked the "I Voted" button.

- Polling Place Search. For those respondents in the two treatment conditions, the site recorded when the respondent clicked the "Find Your Polling Place" link.

- Validated Vote. Respondents who had the same first name, last name, and birthdate as a record in their state's voter file were matched at the group level to allow statistical analysis on the relationship between the treatment and real world behaviour (see below).

Table S2 shows balance tests for the demographic variables. There were no significant differences (all pairwise two-tailed t tests indicated $p > 0.05$) between the treatment and control groups on any of these variables, suggesting that random assignment was successful.

Tables S3 show additional balance tests for the demographic variables of friends. These results show that the user treatment is uncorrelated with the attributes of the people the user is connected to, suggesting that any difference we find between friends of those who received the treatment and friends of those who were in the control group is either due to sampling variation or due to a causal effect of the user treatments on the friends.

	Mean	Min	Max
Age	34.7 (SD 14.8)	18	110
Male	41.3%	0	1
Partisan	0.2%	0	1
Ideologue	0.8%	0	1
Liberal	0.4%	0	1
Conservative	0.5%	0	1
Democrat	0.1%	0	1
Republican	0.1%	0	1
Self-Reported Vote	20.0%	0	1
Polling Place Search	2.4%	0	1
Validated Vote	50.8%	0	1

Table S1. Summary statistics for 61 million Facebook users who logged in on Election Day.

	Social Message		Message		No Message	
Age	34.894	(0.003)	34.907	(0.032)	34.904	(0.032)
Female	58.145%	(0.011%)	58.187 %	(0.106%)	58.255%	(0.106%)
Partisan	0.198%	(0.001%)	0.193%	(0.009%)	0.197%	(0.009%)
Ideologue	0.730%	(0.002%)	0.714%	(0.018%)	0.764%	(0.019%)
Liberal	0.381%	(0.001%)	0.355%	(0.013%)	0.410%	(0.014%)
Conservative	0.397%	(0.001%)	0.410%	(0.014%)	0.413%	(0.014%)
Democrat	0.122%	(0.001%)	0.108%	(0.007%)	0.122%	(0.008%)
Republican	0.088%	(0.001%)	0.099%	(0.007%)	0.088%	(0.006%)

Table S2. Comparison of means across the two message types and the control. Here we show the mean and the standard error. It is important to note that people rarely self-report political characteristics on their Facebook profile (less than 2%, as shown).

	Social Message		Message		No Message	
Age	29.829	(12.752)	29.815	(12.748)	29.829	(12.752)
Female	59.414%	(52.133%)	59.374%	(52.135%)	59.382%	(52.139%)
Partisan	0.207%	(3.153%)	0.206%	(3.150%)	0.205%	(3.149%)
Ideologue	0.866%	(4.819%)	0.863%	(4.811%)	0.865%	(4.811%)
Liberal	0.410%	(6.388%)	0.408%	(6.376%)	0.408%	(6.377%)
Conservative	0.456%	(6.741%)	0.455%	(6.730%)	0.457%	(6.744%)
Democrat	0.106%	(3.464%)	0.105%	(3.451%)	0.105%	(3.445%)
Republican	0.101%	(3.250%)	0.101%	(3.247%)	0.100%	(3.246%)
Number of dyads	8,890,938,491		90,886,141		91,017,926	
Number of users	60,055,176		611,044		613,096	

Table S3. Comparison of means of friend attributes across the ego's two message types and the control group. Here we show the mean and the standard deviation. It is important to note that people rarely self-report political characteristics on their Facebook profile (less than 2%, as shown).

```
# load necessary libraries
library(digest)

# determine the amount of time the program takes to excute
time_start = Sys.time()

# use the SIMyahtzee.R program to determine the number of observations
necessary to achieve a pretermined level of preceision for the
indvidual level estimates of the behavior
required_obs <- 25

# the log file caputes text information about the data processing as
the program excutes
log_file <- "log_file.txt"

# Create a large set of salts to use.  We only have to do this once.
loop_max = 1000
quick.number.to.letter <- function(my_num) {
      if (my_num <= 0) {
            # This function not intended for negative numbers.
            return("A")
      }
      results <- NULL
      while (my_num != 0) {
            remainder = my_num %% 26
            my_num = floor(my_num/26)
            results = paste(results, LETTERS[remainder+1], sep="")
      }
      return(results)
}

salts <- NULL
for (i in 1:loop_max) {
      salts <- c(salts, quick.number.to.letter(i))
}
sum(duplicated(salts)) # Zero

# Set the group size. (we selected 5 for our application but this is an
arbitrary choice.)
# No counts will be taken from groups not of this size.
group_size = 5
loop_count = 1

# Load the destination dataset
destination <- read.csv("DESTINATION_FILE.csv", header=TRUE)

# Remove invalid records.
destination$remove = (is.na(destination$dob_day) |
is.na(destination$dob_month) | is.na(destination$dob_year) |
is.na(destination$first_name) | is.na(destination$last_name))
destination$remove = (destination$remove | destination$dob_day == 0 |
destination$dob_month == 0 | destination$dob_year == 0)
destination$remove = (destination$remove | destination$first_name ==
"NULL" | destination$last_name == "NULL")
```

```
destination <- destination[!destination$remove, 1:ncol(destination)]

# Set the name to uppercase, format date as YYYYMMDD and concatenate
all these to create the input to the hash.
destination$tohash <- paste(toupper(destination$first_name),
toupper(destination$last_name),
format(as.Date(paste(destination$dob_year, "/", destination$dob_month,
"/", destination$dob_day, sep="")),
"%Y%m%d"), sep="")

# Load the orgin file.
orgin <- read.csv("ORIGIN_FILE.csv", header=TRUE)

# Remove invalid origin data records.
orgin$remove = (is.na(orgin$first_name) | is.na(orgin$last_name) |
is.na(orgin$dob) | is.na(orgin$behavior))
orgin <- orgin[!orgin$remove,1:ncol(origin)]

# Set the name to uppercase, format date as YYYYMMDD and concatenate
all these to create the input to the hash.
orgin$tohash <- paste(toupper(orgin$first_name),
toupper(orgin$last_name), orgin$dob, sep="")

### Begin a loop to repeatedly group origin records and use behavioral
frequencies to label destination records ###
start_destination_record_count = nrow(destination) # The termination
condition is >= 99% labeled.
labeled_destination_record_count = 0
while ((loop_count <= loop_max) &
((labeled_destination_record_count/start_destination_record_count) <
.99)) {

write(paste("-----", loop_count, "-----"), file=log_file, append=TRUE,
sep="")
write(paste("Begin iteration", loop_count, "at", Sys.time()),
file=log_file, append=TRUE, sep="")

# Calculate a hash for each row
# Each round, we will add a different salt to the hash.
destination$hash_value = sapply(paste(as.character(destination$tohash),
salts[loop_count], sep=""), digest, algo="sha256", serialize=FALSE)

# Only keep the last 7 hash places.  These values are the largest we
can handle numerically.
destination$hash_value = substr(destination$hash_value, 58, 64)

# The mod value should be the number that will maximize the number of
groups of specified group size
mod_value = round(nrow(orgin)/group_size)

# Mod to get each rows group label
destination$group_label =
as.numeric(as.hexmode(destination$hash_value)) %% mod_value


# Calculate a hash for each row
# Each round, we will add a different salt to the hash.
```

```
orgin$hash_value = sapply(paste(as.character(orgin$tohash),
salts[loop_count], sep=""), digest, algo="sha256", serialize=FALSE)

# Only keep the last 7 hash places.  These values are the largest we
can handle numerically.
orgin$hash_value = substr(orgin$hash_value, 58, 64)

# Group the origin file with the same mod value we used on the
destination file
orgin$group_label = as.numeric(as.hexmode(orgin$hash_value)) %%
mod_value

# Count the records per group and the number of people who exhibit the
behavior per group
ct1 = xtabs( ~ group_label, data=orgin)
group_data = data.frame(as.numeric(names(ct1)), as.numeric(ct1))
names(group_data) = c("group_label", "group_count")

ct2 = xtabs(behavior ~ group_label, data=orgin)
group_data2 = data.frame(as.numeric(names(ct2)), as.numeric(ct2))
names(group_data2) = c("group_label", "group_behavior_count")

# It is a safe assumption that ct1 and ct2 will have the same set of
groups (unless you tell the R program to drop empty groups, which we
don't).
# It is NOT always a safe assumption that ct1 and ct2 will have a
complete list of all possible groups.
# It is possible some potential modulus remainder is not present in the
group of observed hashes.
group_data <- merge(group_data, group_data2)

# Log some statistics concerning group sizes.
write("Group Size summary:", file=log_file, append=TRUE, sep="")
write(names(summary(group_data$group_count)), file=log_file,
append=TRUE, sep="\t", ncolumns=6)
write(summary(group_data$group_count), file=log_file, append=TRUE,
sep="\t", ncolumns=6)

# Find those groups that are of the previously specified group size.
good_groups = (group_data$group_count == 5)

write(paste("We looked for groups of size", group_size), file=log_file,
append=TRUE, sep="")
write(paste("We found", sum(good_groups), "such groups out of",
length(good_groups), "total groups."), file=log_file, append=TRUE,
sep="")
if (sum(good_groups) == 0) {
      # Provide unmatching group label if there are no good groups.
      good_group_values = data.frame(-9999, -1)
} else {
      good_group_values =
data.frame(group_data$group_label[good_groups],
group_data$group_behavior_count[good_groups])
}
names(good_group_values) = c("group_label", paste("yahtzee",
loop_count, sep=""))
```

```
destination <- merge(destination, good_group_values, all.x=TRUE,
all.y=FALSE)

# Remove the group_label and hash_value columns
destination <- destination[,-match("group_label",names(destination))]
destination <- destination[,-match("hash_value",names(destination))]

# Check to see if we have enough obs for each destination record.
destination$filled_columns = rowSums(!is.na(destination)) - 9  # There
are variable columns in the data in our application.  All other are
yahtzees.
labeled_destination_record_count = sum(destination$filled_columns >=
required_obs)

write("Observations per row summary:", file=log_file, append=TRUE,
sep="")
write(names(summary(destination$filled_columns)), file=log_file,
append=TRUE, sep="\t", ncolumns=6)
write(summary(destination$filled_columns), file=log_file, append=TRUE,
sep="\t", ncolumns=6)
write(paste("How many destination records have at least", required_obs,
"obs?", labeled_destination_record_count), file=log_file, append=TRUE,
sep="")
write(paste("What percent of all destination records have at least",
required_obs, "obs?",
labeled_destination_record_count/nrow(destination)),
file=log_file, append=TRUE, sep="")

# Remove the filled_columns column
destination <- destination[,-
match("filled_columns",names(destination))]

loop_count = loop_count + 1
}
### Ended the loop ###

# Write out all the observed behavior counts for each destination
record.
write.table(destination, file="Results.csv", append=TRUE,
col.names=FALSE, row.names=FALSE, sep=",")

time_diff = Sys.time() - time_start
write(print(time_diff), file=log_file, append=TRUE, sep="")
```