S2. Supplemental Matrials Part 2.

Reproducibility in small-N treatment research in aphasia and related disorders: a tutorial

Introduction

This document details the code batch calculate effect sizes.

Setup

Load packages

```
library(here)           # for locating files
library(GGally)         # Plotting
library(SingleCaseES)   # calculating SMD, Tau-U
library(lme4)           # frequentist mixed-effects models
library(emmeans)        # estimating effect sizes from lme4
library(brms)           # bayesian mixed-effects models
library(tidybayes)      # estimating effect sizes from brms
library(ggdist)         # Visualizing posterior distributions
library(tidyverse)      # data wrangling and plotting

# set a seed for reproducibility
set.seed(42)
```

Load effect size functions

For this analysis, we created a number of custom effect size functions. Functions serve to isolate complex code that serves a specific purpose from code used in the primary analysis. These functions are in in many way specific to the present data set, but were created to be generalizable to other similar data. The functions are in a file called `effect-size-functions.R` and are highly commented to explain each step.

```
# load functions that batch calculate effect sizes
source(here("R", "effect-size-functions.R"))
# print the four functions
ls()
```

```
## [1] "glmmES"     "hook_chunk" "PMG"        "SMD_br"     "Tau_custom"
```

Read in data

Note that the current setup uses RStudio R projects (https://support.rstudio.com/hc/en-us/articles/200526207-Using-RStudio-Projects). One of the features of R projects is that the working directory is automatically set to the project root (the folder with the .Rproj). A discussion of R projects can be found at https://www.tidyverse.org/blog/2017/12/workflow-vs-script/. In this case `here("study-data")` refers to the /study-data folder inside the project.

```
# create a list of files
files <- list.files(
                 here("study-data"), # look in the study-data folder
                 full.names = TRUE,  # use the full paths of the files
                 pattern = ".csv",   # only read in .csv files
                 recursive = TRUE)   # include files within subfolders

# read in the files and combine them together
# map_df takes a function, in this case read_csv().
# show_col_types suppresses output sinc we're reading in many files
df <- files %>%
  map_dfr(read_csv, show_col_types = FALSE)
```

Calculate effect sizes

d_{BR}

The following calculates d_{BR} using the observations used by Wambaugh et al., (2017)

```
# start with all item-level data
df_smd = df %>%
  # filter for only baseline and treatment phases included in
  # wambaugh 2017 smd calculation
  filter(spt2017 == "pre" | spt2017 == "post" ) %>%
  # for each combination of these variables
  group_by(participant, phase, condition, itemType, session,
           spt2017) %>%
  # calculate the number of correct responses
  # this ends with the number of correct responses per session
  # per participant, condition, and itemType
  summarize(correct = sum(response), .groups = "drop") %>%
  group_by(participant, condition, itemType) %>%
  summarize(SMD_br(outcome = correct, phase = spt2017,
               bl_phase = "pre", tx_phase = "post"), .groups = "drop")
```

The following calculates d_{BR} using all baseline observations

```r
# start with all item-level data
df_smd_all = df %>%
  # filter for only baseline and treatment phases included in
  # wambaugh 2017 smd calculation
  filter(phase == "baseline" | phase == "treatment" & spt2017 == "post") %>%
  # for each combination of these variables
  group_by(participant, phase, condition, itemType, session,
           spt2017) %>%
  # calculate the number of correct responses
  # this ends with the number of correct responses per session
  # per participant, condition, and itemType
  summarize(correct = sum(response), .groups = "drop") %>%
  group_by(participant, condition, itemType) %>%
  summarize(SMD_br(outcome = correct, phase = phase,
                bl_phase = "baseline", tx_phase = "treatment"), .groups = "drop")
```

The following calculates d_{BR} for each phoneme, and then averaging the two scores, using the last 5 baseline observations

```r
df_smd_phoneme = df %>%
  # filter for only baseline and treatment phases included in
  # wambaugh 2017 smd calculation
  filter(spt2017 == "pre" | spt2017 == "post" ) %>%
  # for each combination of these variables
  group_by(participant, phase, condition, itemType, session,
           spt2017, phoneme) %>%
  # calculate the number of correct responses
  # this ends with the number of correct responses per session
  # per participant, condition, and itemType
  summarize(correct = sum(response), .groups = "drop") %>%
  group_by(participant, condition, itemType, phoneme) %>%
  summarize(SMD_br(outcome = correct, phase = spt2017,
                bl_phase = "pre", tx_phase = "post"), .groups = "drop")

# when one phoneme has no variability in the baseline phase, use the variability
# from the other phoneme
df_smd_phoneme_fix = df_smd_phoneme %>%
  group_by(participant, condition, itemType) %>%
  # only keep rows where there is an NA value (within the grouping)
  filter(any(is.na(SMD))) %>%
  # get rid of rows where all values within the grouping is NA
  filter(!all(is.na(SMD))) %>%
  # for each grouping pair, set the NA to the other phoneme SD
  # then recalculate SMD if SMD is NA
  mutate(sd = max(sd, na.rm = T),
         SMD = ifelse(is.na(SMD), change/sd, SMD)
         ) %>%
  ungroup()


# update the original data using the fixed data
df_smd_phoneme = df_smd_phoneme %>%
```

```
  rows_update(df_smd_phoneme_fix,
              by = c("participant", "condition", "itemType", "phoneme")) %>%
  group_by(participant, condition, itemType) %>%
  # summarize the smd and sd across phonemes
  # note whether or not the sd was used from the other phoneme
  # incases where variability was zero
  summarize(SMD = mean(SMD, na.rm = T),
            sd = mean(sd, na.rm = T),
            imputed = ifelse(any(!is.na(note)), 1, 0),
                      .groups = "drop")
```

The following calculates d_{BR} for each phoneme, and then averaging the two scores, using the last all baseline observations

```
df_smd_phoneme_all = df %>%
  # filter for only baseline and treatment phases included in
  # wambaugh 2017 smd calculation
  filter(phase == "baseline" | phase == "treatment" & spt2017 == "post") %>%
  # for each combination of these variables
  group_by(participant, phase, condition, itemType, session,
           spt2017, phoneme) %>%
  # calculate the number of correct responses
  # this ends with the number of correct responses per session
  # per participant, condition, and itemType
  summarize(correct = sum(response), .groups = "drop") %>%
  group_by(participant, condition, itemType, phoneme) %>%
  summarize(SMD_br(outcome = correct, phase = phase,
              bl_phase = "baseline", tx_phase = "treatment"), .groups = "drop")

# when one phoneme has no variability in the baseline phase, use the variability
# from the other phoneme
df_smd_phoneme_all_fix = df_smd_phoneme_all %>%
  group_by(participant, condition, itemType) %>%
  # only keep rows where there is an NA value (within the grouping)
  filter(any(is.na(SMD))) %>%
  # get rid of rows where all values within the grouping is NA
  filter(!all(is.na(SMD))) %>%
  # for each grouping pair, set the NA to the other phoneme SD
  # then recalculate SMD if SMD is NA
  mutate(sd = max(sd, na.rm = T),
         SMD = ifelse(is.na(SMD), change/sd, SMD)) %>%
  ungroup()

# update the original data using the fixed data
df_smd_phoneme_all = df_smd_phoneme_all %>%
  rows_update(df_smd_phoneme_all_fix,
              by = c("participant", "condition", "itemType", "phoneme")) %>%
  group_by(participant, condition, itemType) %>%
  summarize(SMD = mean(SMD, na.rm = T),
            sd = mean(sd, na.rm = T),
            imputed = ifelse(any(!is.na(note)), 1, 0),
            .groups = "drop")
```

PMG

The following calculates PMG using the observations used by Wambaugh et al., (2017)

```
df_pmg = df %>%
  # This uses the observations used in wambaugh et al., 2017
  filter(spt2017 == "pre" | spt2017 == "post" ) %>%
  group_by(participant, phase, condition, itemType, session,
           spt2017, trials) %>%
  # calculate the number of correct responses
  # this ends with the number of correct responses per session
  # per participant, condition, and itemType
  # also calculate the number of trials which is necessary for PMG
  summarize(correct = sum(response),
            trials = unique(trials)*2, .groups = "drop") %>%
  group_by(participant, condition, itemType) %>%
  summarize(PMG(outcome = correct, phase = spt2017, nitems = trials,
                bl_phase = "pre", tx_phase = "post"), .groups = "drop")
```

The following calculates PMG using all baseline observations

```
df_pmg_all = df %>%
  filter(phase == "baseline" | phase == "treatment" & spt2017 == "post") %>%
  group_by(participant, phase, condition, itemType, session,
           spt2017, trials) %>%
  # calculate the number of correct responses
  # this ends with the number of correct responses per session
  # per participant, condition, and itemType
  # also calculate the number of trials which is necessary for PMG
  summarize(correct = sum(response),
            trials = unique(trials)*2, .groups = "drop") %>%
  group_by(participant, condition, itemType) %>%
  summarize(PMG(outcome = correct, phase = phase, nitems = trials,
                bl_phase = "baseline", tx_phase = "treatment"), .groups = "drop")
```

Tau-U

```
df_tau = df %>%
  # filter for all obseravtions in the baseline and
  # treatment phase.
  filter(phase == "baseline" | phase == "treatment") %>%
  group_by(participant, phase, condition, itemType, session) %>%
  # add up the number of correct responses
  summarize(correct = sum(response), .groups = "drop") %>%
  group_by(participant, condition, itemType) %>%
  # This calls a custom Tau function which uses Tau-U
  # in cases where the trend is >= the cutoff value
  summarize(Tau_custom(outcome = correct, phase = phase,
                       bl_phase = "baseline", tx_phase = "treatment",
                       session = session, cutoff = 0.33), .groups = "drop")
```

Bayesian Mixed-effects models

Setup data

```
df_itts_group = df %>%
  filter(phase == "baseline" | phase == "treatment") %>%
  mutate(baseline_slope = session,
         level_change = ifelse(phase == "baseline", 0, 1),
         slope_change = (session - (n_baselines+2))*level_change) %>%
  select(response, participant, condition, itemType, phase,
         phoneme, item, baseline_slope, level_change, slope_change)
```

Note that we would typically use n_baselines+1 if we sampled performance at every treatment session

Blocked Treated

```
  mod_tx_bl <- brm(
    # outcome variable response
    # the 0 a+ Intercept syntax llows us to put a non-centered prior on the intercept
    # the population-level effects (fixed effects in frequentist terminology)
    # are baseline_slope, level_change, and slope_change. The group-level
    # effects (random effects) are in parentheses.
    response ~ 0 + Intercept + baseline_slope + level_change + slope_change +
               (1 + baseline_slope + level_change + slope_change | participant) +
               (1| item),
            # data, filtered for the itemType and condition
             data = df_itts_group %>% filter(condition == "blocked",
                                             itemType == "tx"),
             family = bernoulli(), # special case of binomial with 1 trial
             iter = 3000, # number of iterations
             warmup = 1000, # number of iterations to toss
             cores = 4, chains = 4, # 4 Markov chains across 4 computer cores
             # prior distributions include a specific prior on the intercept
             # and a general prior on the population-level effects
             prior = c(
               prior(normal(-1, 2.5), class = b, coef = Intercept),
               prior(normal(0, 2.5), class = b)
             ),
            # because of divergent transitions see
            # cran.r-project.org/web/packages/brms/vignettes/brms_overview.pdf
             control = list(adapt_delta = 0.9),
            # set a seed for reproducibility
             seed = 42,
            # save the model so we don't have to refit it every time we compile
             file = "models/mod_tx_bl",
            # only refit the model when something changes
             file_refit = "on_change"
  )
```

Blocked Generalization

```
mod_gx_bl <- brm(
  response ~ 0 + Intercept + baseline_slope + level_change + slope_change +
             (1 + baseline_slope + level_change + slope_change | participant) +
             (1| item),
                 data = df_itts_group %>% filter(condition == "blocked",
                                                  itemType == "gx"),
                 family = bernoulli(),
                 iter = 3000,
                 warmup = 1000,
                 cores = 4, chains = 4,
                 prior = c(
                   prior(normal(-1, 2.5), class = b, coef = Intercept),
                   prior(normal(0, 2.5), class = b)
                 ),
                 control = list(adapt_delta = 0.85),
                 seed = 42,
                 file = "models/mod_gx_bl",
                 file_refit = "on_change"
)
```

Random Treated

```
mod_tx_ra <- brm(
  response ~ 0 + Intercept + baseline_slope + level_change + slope_change +
             (1 + baseline_slope + level_change + slope_change | participant) +
             (1| item),
                 data = df_itts_group %>% filter(condition == "random",
                                                  itemType == "tx"),
                 family = bernoulli(),
                 iter = 3000,
                 warmup = 1000,
                 cores = 4, chains = 4,
                 prior = c(
                   prior(normal(-1, 2.5), class = b, coef = Intercept),
                   prior(normal(0, 2.5), class = b)
                 ),
                 seed = 42,
                 control = list(adapt_delta = 0.85),
                 file = "models/mod_tx_ra",
                 file_refit = "on_change"
)
```

Random Generalization

```
mod_gx_ra <- brm(
  response ~ 0 + Intercept + baseline_slope + level_change + slope_change +
             (1 + baseline_slope + level_change + slope_change | participant) +
```

```
               (1| item),
                   data = df_itts_group %>% filter(condition == "random",
                                                   itemType == "gx"),
                   family = bernoulli(),
                   iter = 3000,
                   warmup = 1000,
                   cores = 4, chains = 4,
                   control = list(adapt_delta = 0.9),
                   prior = c(
                     prior(normal(-1, 2), class = b, coef = Intercept),
                     prior(normal(0, 2), class = b)
                   ),
                   seed = 42,
                   file = "models/mod_gx_ra",
                   file_refit = "on_change"
  )
```

Calculate effect sizes for each model. The function takes as arguments the model object the itemtype and condition.

```
es_tx_bl = glmmES(mod_tx_bl, "tx", "blocked")
es_tx_ra = glmmES(mod_tx_ra, "tx", "random")
es_gx_bl = glmmES(mod_gx_bl, "gx", "blocked")
es_gx_ra = glmmES(mod_gx_ra, "gx", "random")
```

Pull the effect sizes together to create a correlation plot

```
# select only the necessary columns
smd =
  df_smd %>%
  select(participant, condition, itemType, SMD, sd)
# select only the necessary columns
pmg =
  df_pmg %>%
  select(participant, condition, itemType, PMG, raw_change = raw_change_exit, baseline_score)
# select only the necessary columns
tau =
  df_tau %>%
  select(participant, condition, itemType, Tau = Est)
# combine the effect sizes from each of the conditions
# from the bayesian models then
# select only the necessary columns
bglmm =
  bind_rows(es_tx_bl, es_tx_ra, es_gx_bl, es_gx_ra) %>%
  select(participant, ES, unit, itemType, condition) %>%
  pivot_wider(names_from = unit, values_from = ES) %>%
  rename(glmm_logit = logit, glmm_percent = percent)
# join all the effect sizes together
es = smd %>%
```

```
left_join(pmg, by = c("participant", "itemType", "condition")) %>%
left_join(tau, by = c("participant", "itemType", "condition")) %>%
left_join(bglmm, by = c("participant", "itemType", "condition")) %>%
mutate(itemType = factor(itemType, levels = c("tx", "gx"))) %>%
select(participant, condition, itemType, SMD, PMG, Tau, glmm_logit, glmm_percent)
```

Plot comparisons

This is figure 3. in the manuscript. Code hidden due to length.