Replication Package:

Drivers of COVID-19 Protest Across Localities in Israel: A Machine-Learning Approach

Nina Schlager, Karsten Donnay, Hyunjung Kim and Ravi Bhavnani

This file contains replication information for the analysis performed in Schlager et al. (2023). *'Drivers of COVID-19 Protest Across Localities in Israel: A Machine-Learning Approach'*, *Political Research Exchange* DOI: 10.1080/2474736X.2023.2257368.

---

1. Data

1.1 Locality characteristics

Israel has one sub-national level of government, i.e., local government, including municipalities, local and regional councils. In this study we focus on municipalities, excluding local and regional councils. This ensures that differences in local authority do not affect our matching of protest events to local governments. Regional and local councils usually cover much larger geographical areas, which ultimately results in upward-biased protest counts relative to smaller authorities. Units of analysis in this study are municipality-weeks. Note that we also more generally refer to municipalities as localities throughout the paper and analysis.

Annual survey data on local government characteristics are available via the Israeli Central Bureau for Statistics (CBS) and provided in `raw_covar.fixed.csv`. `covar.fixed.csv` includes the prepared data on municipality characteristics.

```
# get data
df <- read.csv("data/raw_covar.fixed.csv")

#### re-code and clean-up

# set "-" and ".." to NA
df <- df%>%
  mutate_if(is.character, list(~na_if(., "..")))
df <- df %>%
  mutate_if(is.character, list(~na_if(., "-")))

# covert indicators to numeric
df$locality_code <- as.factor(df$locality_code)
df$locality_id <- as.factor(df$locality_id)
df$locality_name <- as.factor(df$locality_name)

df <- df %>%
  mutate_if(is.character, as.numeric)

# log population size
df$population.count <- log(df$population.count +1)
```

```r
# population rate
df <- df %>%
  dplyr::rename(population.rate_jewish = "population.jewish",
                population.rate_arabs = "population.arabs")

df[is.na(df$population.rate_arabs),]$population.rate_arabs <- 0
df[is.na(df$population.rate_jewish),]$population.rate_jewish <- 0
df[is.na(df$religious.arab_muslim),]$religious.arab_muslim <- 0
df[is.na(df$religious.arab_christian),]$religious.arab_christian <- 0
df[is.na(df$religious.arab_druze),]$religious.arab_druze <- 0

# religious.arab_muslim == 1 if +50% Arab population and +50% Muslim
df$religious.arab_muslim <- as.factor(ifelse(
                              df$population.rate_arabs >= 50 &
                              df$religious.arab_muslim >= 50,
                              "true",
                              "false"))

# religious.arab_christian == 1 if +50% Arab population and +50% Christian
df$religious.arab_christian <- as.factor(ifelse(
                              df$population.rate_arabs >= 50 &
                              df$religious.arab_christian >= 50,
                              "true",
                              "false"))

# religious.arab_druze == 1 if +50% Arab population and +50% Druze
df$religious.arab_druze <- as.factor(ifelse(
                              df$population.rate_arabs >= 50 &
                              df$religious.arab_druze >= 50,
                              "true",
                              "false"))

# education indicators == 0 if NA
df[is.na(df$education.highschools_count),]$education.highschools_count <- 0
df[is.na(df$education.highschool_student_count),]$education.highschool_student_count <- 0

df$locality_code <- as.character(df$locality_code)
df$locality_id <- as.character(df$locality_id)
```

We add an additional binary variable to capture 7 municipalities with an absolute majority of Haredi (ultra-Orthodox) Jews, as coded in Gal et al. (2017): Social Service Budgeting in Israeli Local Authorities.

```r
# add new religious.jewish_orthodox variable
df$religious.jewish_orthodox <- "false"

df[df$locality_code == "6100" # Bnei Brak
   | df$locality_code == "3780" # Betar Illit
   | df$locality_code == "3797" # Modi'in Illit
   | df$locality_code == "3660" # Immanu'el
   | df$locality_code ==  "2600" # Elat
   | df$locality_code == "1137" # Qiryat Ye'arim
   | df$locality_code == "922" ,]$religious.jewish_orthodox <- "true" #Rekhasim

df$religious.jewish_orthodox <- as.factor(df$religious.jewish_orthodox)
```

```r
# order
df <- df[,c(1:7,56,8:55)]

covar.fixed <- df

# optional: save data
#write.csv(df,"data/covar.fixed.csv", row.names = FALSE)
```

1.2 Protest events

To effectively disentangle correlates of protests directly related to COVID-19, we combined ACLED's curated COVID-19 disorder tracker data (`raw_acled.covid.xlsx`) with non-Covid disorder events in Israel and Palestine (`raw_acled.non_covid.csv`). The merged protest event data is provided in `protest.event.csv`.

`Pop_Sex_Age_Relig_Setl.shp` in the sub-directory `data/shp_cbs` includes spatial information of Israeli local governments. We subset the shape file to our study sites, i.e. the 189 municipality polygons, and assign protest events to municipalities, if their point coordinates overlap with the municipality polygon area.

```r
#### import spatial data
dfshp <- st_read("data/shp_cbs/Pop_Sex_Age_Relig_Setl.shp")

# rename locality_code
dfshp <- dfshp %>%
  dplyr::rename(locality_code = "LocalityCo")

dfshp$locality_code <- as.character(dfshp$locality_code)

# subset
dfshp <- merge(dfshp, covar.fixed, by= c("locality_code"), all.y = TRUE)

# fix crs
st_crs(dfshp)
dfshp <- st_transform(dfshp, 4326)

#### convert protest events to spatial object
df <- st_as_sf(df,
               coords=c("longitude", "latitude"),
               crs = 4326)

#### intersect protest events with localities
dfgeo <- st_intersection(df, dfshp)

# get long and lat, then drop geometry information
dfgeo <- dfgeo %>%
  dplyr::mutate(longitude = sf::st_coordinates(.)[,1],
                latitude = sf::st_coordinates(.)[,2])

df <- dfgeo %>% st_drop_geometry(dfgeo)

# subset to relevant
df <- df[c("week_id", "locality_code", "locality_id", "locality_name",
           "protest_type", "protest_sub_type", "actor1",
           "notes", "longitude", "latitude", "date")]
```

```r
df$locality_id <- as.character(df$locality_id)

protest.event <- df

# optional: save data
#write.csv(df, "data/protest.event.csv", row.names = FALSE)
```

We then use the variable ***locality_id*** as an identifier and aggregate to municipality-weeks by counting the number of protests, also per type, and merge this with municipality characteristics.

```r
# list combinations of municipalities and weeks
obs <- tibble(
  locality_id = unique(covar.fixed$locality_id),
  week_id = c(10:135, 10:72))

obs <- obs  %>% complete(locality_id, week_id)

# count protest events per municipality-week
count <- df %>%
  group_by(locality_id, week_id) %>%
  dplyr::summarise(protest_all_count = length(locality_id))

# merge with all locality-weeks
count <- merge(obs, count, by= c("week_id", "locality_id"),
               all.x = TRUE)

# count covid protest events per municipality-week
count_cov <- df %>%
  filter(protest_type == "covid19") %>%
  group_by(locality_id, week_id) %>%
  dplyr::summarise(protest_covid_count = length(locality_id))

# merge with all locality-weeks
count <- merge(count, count_cov, by= c("week_id", "locality_id"),
               all.x = TRUE)

# count other protest events per municipality-week
count_other <- df %>%
  filter(protest_type == "non_covid") %>%
  group_by(locality_id, week_id) %>%
  dplyr::summarise(protest_non_covid_count = length(locality_id))

# merge with all locality-weeks
count <- merge(count, count_other, by= c("week_id", "locality_id"),
               all.x = TRUE)

# replace NA protest counts with 0
df <- count %>% replace(is.na(.), 0)

# merge protest events with municipality covariates
df <- merge(df, covar.fixed, by= c("locality_id"), all.x = TRUE)
```

1.3 Covid-19 infection dynamics

We also add weekly information on cumulative Covid-19 cases, test and hospitalizations per municipality. The raw data provided by the Israeli Ministry of Health is available in `raw_covar.covid19.csv`. The constructed data is saved as `covar.covid19.csv`.

```
# get data
covid <- read.csv("data/raw_covar.covid19.csv")

# re-name
covid <- covid %>%
  dplyr::rename(locality_code = "town_code",
                covid19.cases = "accumulated_cases",
                covid19.hospitalizations = "accumulated_hospitalized",
                covid19.tests = "accumulated_diagnostic_tests",
                covid19.vaccine = "accumulated_vaccination_first_dose")

# subset to relevant columns
covid <- covid[, c(1, 4, 5, 9, 13, 15)]

# subset to study period
covid <- subset(covid, covid$date < "2022-07-28")

# get calendar week from date
covid$week <- strftime(covid$date, format = "%Y-%V")
unique(covid$week)

# create week_id starting from calendar week 11, 2020
covar.covid19 <- data.frame(covid %>%
                      group_by(week) %>%
                      dplyr::mutate(week_id = cur_group_id()+10))

# check
min(covar.covid19$week_id)
max(covar.covid19$week_id)

# get cumulative counts per municipality-week
covar.covid19 <- covar.covid19 %>%
  mutate_if(is.character, as.numeric)  %>%
  group_by(locality_code, week_id) %>%
  slice_tail(n = 1)

# order
covar.covid19 <- data.frame(covar.covid19)
covar.covid19 <- covar.covid19[, c(8,1,3:5)]
covar.covid19$locality_code <- as.character(covar.covid19$locality_code)
covar.covid19$week_id <- as.character(covar.covid19$week_id)

# optional: save data
#write.csv(covar.covid19,"data/covar.covid19.csv", row.names = FALSE)

# merge with other covariates
df <- merge(df, covar.covid19, by= c("week_id", "locality_code"), all.x = TRUE)
```

1.4 Covid-19 policy responses

Lastly, we add information on government policy responses to the pandemic sourced from the Oxford Covid-19 Government Response Tracker (OxCGRT). We aggregate daily observations to weekly means and merge with the other covariates. The raw data is available in `raw_covar.oxford.xlsx` and the constructed data is saved as `covar.oxford.csv`.

```
# get data
oxford <- read_excel("data/raw_covar.oxford.xlsx")

# reshape to long format
oxford <- reshape2::melt(oxford, na.rm = FALSE, id = c("index"))
names(oxford) <- c("index","date", "value")

# get dates
oxford$date <- anydate(oxford$date)

# subset to study period
oxford <- subset(oxford, oxford$date >= "2020-03-01")
oxford <- subset(oxford, oxford$date <= "2022-07-28")

# get calendar week from date
oxford$week <- strftime(oxford$date, format = "%Y-%V")
unique(oxford$week)

# create week id,
oxford <- data.frame(oxford %>%
                     group_by(week) %>%
                     dplyr::mutate(week_id = cur_group_id()+9))

min(oxford$week_id)
max(oxford$week_id)

# subset to relevant and reshape
oxford <- oxford[, c(5, 1, 3)]
oxford <- reshape(oxford, idvar = "week_id", timevar = "index",
                  direction = "wide")

names(oxford) <- c("week_id","covid19.gov_stringency_index",
                   "covid19.economic_support_index",
                   "covid19.containment_health_index",
                   "covid19.gov_response_index")

# optional: save data
#write.csv(oxford,"data/covar.oxford.csv", row.names = FALSE)

# merge with covariate data
df <- merge(df, oxford, by= c("week_id"), all.x = TRUE)

# drop locality_code and locality_name
df <- df[, c(-2,-7)]

# check
summary(df)
```

```r
# optional: save data
#write.csv(df,"data/protest.locality-week.csv", row.names = FALSE)

df_save <- df
```

Figure 1

We use the spatial information provided in `Pop_Sex_Age_Relig_Setl.shp` in the sub-directory `data/shp_cbs` to plot a map of Israel. Figure 1 shows the the number of protests between March 2020 and July 2022 in Israeli municipalities.

```r
#### protest events

# count protest events per municipality
count <- protest.event %>%
  group_by(locality_id) %>%
  dplyr::summarise(count = length(protest_type),
                   locality_id = paste0(locality_id)) %>%
  slice_head(n = 1)

# get mean number of protests across 189 municipalities
sum(count$count)/189

# merge count back to protest.event to get point coordinates
dfpoint <- merge(count, protest.event, by = "locality_id")

# subset to one observation per municipality
dfpoint <- dfpoint %>%
  group_by(locality_id) %>%
  filter(row_number(locality_id) == 1)

#### municipalities

# get spatial data
dfgeo <- st_read("data/shp_cbs/Pop_Sex_Age_Relig_Setl.shp")

# fix crs
st_crs(dfgeo)
dfgeo <- st_transform(dfgeo, 4326)

#### neighboring countries

# get spatial info of neighboring countries
world <- ne_countries(scale = "large", returnclass = "sf")
some.countries <- c("Jordan", "Lebanon", "Syria", "Egypt")
some.maps <- map_data("world", region = some.countries)
some.israel <- c("Israel")
israel <- map_data("world", region = some.israel)

# convert to sf
region.data <- st_as_sf(some.maps, coords = c("long", "lat"),
                        remove = FALSE,
                        crs = 4326, agr = "constant")
```

```r
# define coordinates for country labels
region.lab.data <- data.frame(lat = c(30.590882, 31.737753,
                                      33.548499, 33.015812),
                              long = c(33.782909, 36.040599,
                                       35.625866, 36.548717),
                              name = c("Egypt", "Jordan",
                                       "Lebanon", "Syria"))

region.lab.data$name <- as.character(region.lab.data$name)

# convert to sfx
region.lab.data <- st_as_sf(region.lab.data,
                            coords = c("long", "lat"),
                            remove = FALSE,
                            crs = 4326,
                            agr = "constant")

# plot options
legend_size <- c(2,4,6,8)

#### plot

fig1 = ggplot() +
  geom_sf(data = world,
          fill = "#eeeff0", colour = "black", alpha= .2, linewidth = 0.2) +
  geom_text(data = region.lab.data,
            aes(x = long, y = lat, label = name), size = 4) +
  geom_sf(data = dfgeo,
          aes(fill = "#eeeff0"), alpha= .4, show.legend = FALSE) +
  geom_point(data = dfpoint,
             aes(x = longitude, y = latitude, size = count,
             colour = count), alpha = .8) +
  scale_fill_manual(name = "",
                    values = c("#eeeff0", "#eeeff0"),
                    labels = c("study site", "NA")) +
  guides(colour=guide_legend(override.aes = list(size = legend_size),
                             title.position = "top", nrow = 2),
           size = FALSE) +
  scale_color_continuous(low ="#ff6699", high = "#ff6699",
                         breaks =  c(1, 25, 50, 100),
                         labels = c("1 - 25", "26 - 50",
                                    "51 - 100", " > 100"),
                         name = "Protest events per municipality") +
  scale_size(range = c(1, 8)) +
  annotation_scale(location = "bl", width_hint = 0.25) +
  xlim(33.25, 37.25) +
  ylim(29.5, 33.75) +
  theme_void() +
  theme(legend.position="left",
        legend.box = "vertical",
        legend.title = element_text(size=12, face="bold"),
        legend.text = element_text(size=12))
fig1
```

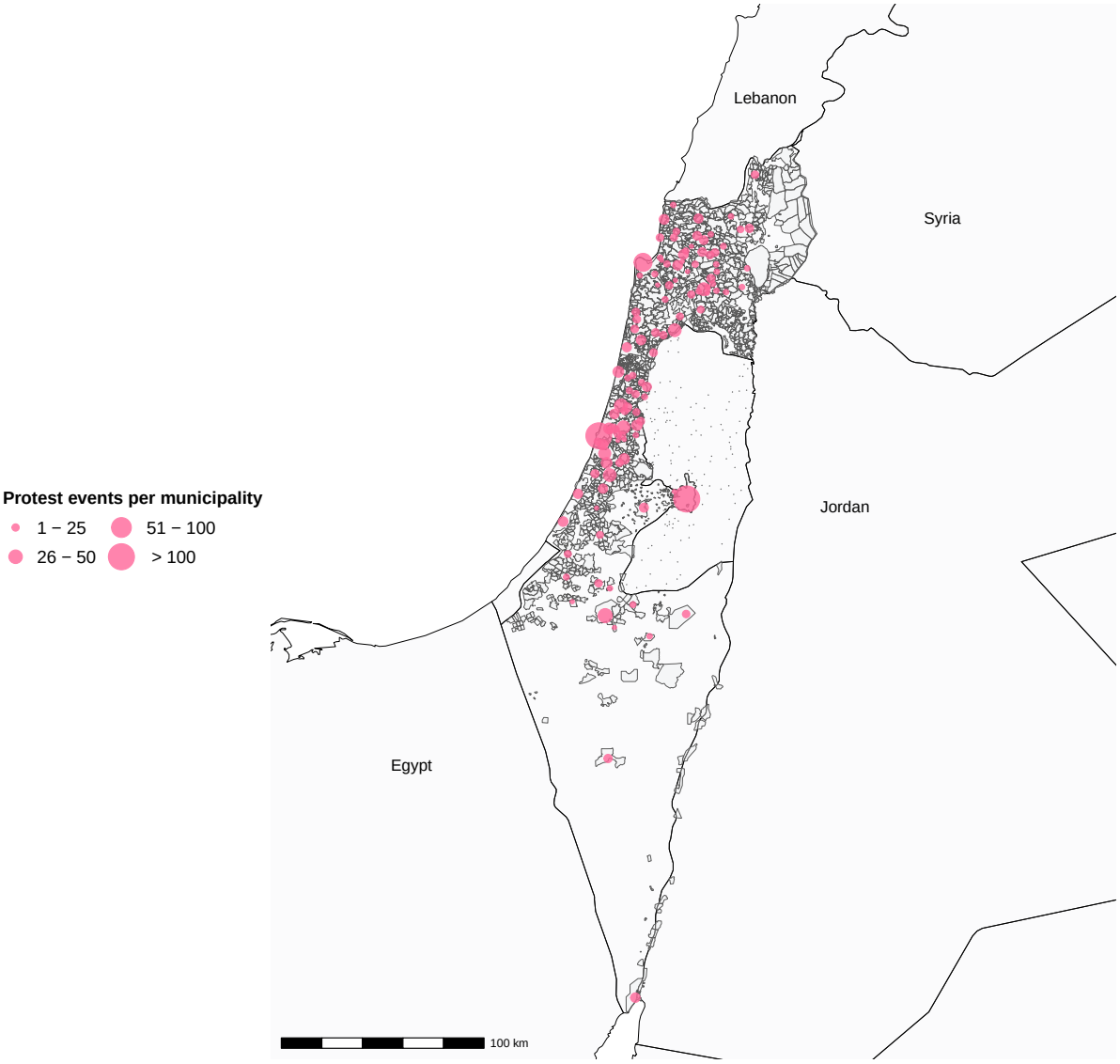

Figure 1: Protest counts per municipality between March 2020 and July 2022 (Figure 1).

```
# optional: save Figure 1 as .svg or .pdf
#ggsave("figures/figure_1.svg", fig1,
#  width=16.79,height=10.65,units="in",dpi=300)
#ggsave("figures/figure_1.pdf", fig1,
#  width=16.79,height=10.65,units="in",dpi=300)
```

Figure 2

Figure 2 displays the number of daily Covid (pink) and non-Covid protests (purple) in Israeli municipalities relative to the daily change rate of Covid cases (red) and vaccinations (blue) between March 2020 and July 2022. Daily information on Covid dynamics is featured in `covar.covid19-figure.csv`.

```
# get protest data
df <- protest.event
df$protest_type <-as.factor(df$protest_type)
df$date <- as.Date(df$date)

# count protest events per type and day
df2cov <- df %>%
  filter(protest_type == "covid19") %>%
  group_by(date) %>%
  dplyr::mutate(count_covid = length(locality_id))

df2 <- df %>%
  filter(protest_type == "non_covid") %>%
  group_by(date) %>%
  dplyr::mutate(count_other = length(locality_id))

df2all <- df %>%
  group_by(date) %>%
  dplyr::mutate(count_all = length(locality_id))

# get Covid dynamics
covid19 <- covid
covid19$covid19.cases <- as.numeric(covid19$covid19.cases)
covid19$covid19.vaccine <- as.numeric(covid19$covid19.vaccine)
covid19$locality_code <- as.factor(covid19$locality_code)
covid19$date <- as.Date(covid19$date)

# change
colnames(covid19)[3:6] = c("cases", "hosp", "tests", "vaccine")

# arrange by locality_code date
covid19 <- covid19 %>%
  arrange(locality_code, date)

# fill NA with last value
covid19$cases <- na.locf(covid19$cases, fromLast = TRUE)
covid19$vaccine <- na.locf(covid19$vaccine,fromLast = TRUE)

# mean across localities
covid19 <- aggregate(cbind(cases, vaccine) ~ date,
                     data=covid19, FUN=mean)
```

```r
# computing relative change in % compared to previous day
covid19 <- covid19 %>%
  arrange(date) %>%
  mutate(diff_cases =  (cases - lag(cases))/lag(cases) * 100,
         diff_vaccine = (vaccine - lag(vaccine))/lag(vaccine) * 100)

# subset
covid19 <- data.frame(covid19)

# fill NA with last value
covid19$diff_cases <- na.locf(covid19$diff_cases, fromLast = TRUE)
covid19$diff_vaccine <- na.locf(covid19$diff_vaccine,fromLast = TRUE)

# set non-finite to 0
setDT(covid19)
covid19[!is.finite(covid19$diff_cases),]$diff_cases<-0
covid19[!is.finite(covid19$diff_vaccine),]$diff_vaccine<-0

# reshape data
dfplot3 <- reshape2::melt(covid19, id.vars=c("date"),
                          measure.vars=c("diff_cases",
                                         "diff_vaccine"))
# prepare % for plotting
dfplot3 <- dfplot3[dfplot3$value <= 100,]

### plot

fig2 <- ggplot2::ggplot(covid19,aes(x=date, fill="locality_code")) +
          stat_smooth(data=dfplot3,
                    aes(y=value, fill=variable), geom = "area",
                    method = "loess", position = "identity", span = .01,
                    color="black", linewidth= .1, alpha=.85) +
          geom_smooth(data = df2cov,
                    aes(x = date, y = count_covid, color = "#ffb3de"),
                    method = "gam", span = .1, fill = "#ffb3de",
                    alpha=.25, linewidth= .35, na.rm = TRUE) +
          geom_smooth(data = df2,
                    aes(x = date, y = count_other, color = "#6048c0"),
                    method = "gam", span = .01, fill = "#6048c0",
                    alpha=.25, linewidth= .35, na.rm = TRUE) +
          scale_fill_manual(name="Covid-19",
                    breaks=c("diff_cases","diff_vaccine"),
                    labels=c("cases","vaccinations"),
                    values=c("#ff6161","#a3a9fd")) +
          guides(colour=guide_legend(
                    override.aes = list(size = 3),
                    title.position = "left",
                    nrow = 1), size = FALSE) +
          scale_color_manual(labels = c("non-Covid", "Covid"),
                    name = "Protest type" ,
                    values = c("#6048c0","#ffb3de")) +
          scale_y_continuous(breaks = c(20*0.05,20*0.10,20*0.15,
                                        20*0.20,20*0.25,20*0.3,
```

```
                                    20*0.4,
                                    20*0.5),
                          limits = c(-3,10),
                          labels=c("5", "10", "15", "20",
                                   "25", "30", "40","50"),
          sec.axis = sec_axis(~ .*5, name ="Protests per day")) +
          scale_x_date(breaks=c(seq.Date(min(dfplot3$date),
                                    max(dfplot3$date),
                                    by="quarter"),
                                    as.Date("2020-03-01")),
                       labels=format(c(seq.Date(min(dfplot3$date),
                                                max(dfplot3$date),
                                                by="quarter"),
                                                as.Date("2020-03-01")),
                                                "%b %Y"),
                       expand = c(0, 0.02)) +
          theme(panel.grid.major = element_blank(),
            panel.grid.minor = element_blank(),
            panel.background = element_blank(),
            axis.text.x = element_text(size =10),
            axis.title.x = element_text(size =12, face = "bold"),
            axis.text.y =  element_text(size =10),
            axis.title.y = element_text(size =12, face = "bold"),
            legend.position="bottom", legend.box = "horizontal",
            legend.title = element_text(size=12, face="bold"),
            legend.text = element_text(size=12)) +
          labs(y="Relative change rate per day (%)", x="")

fig2
```

```
# optional: save Figure 2 as .svg or .pdf
#ggsave("figures/fig_2.svg", fig2,
#  width=16.79,height=10.65,units="in",dpi=300)
#ggsave("figures/fig_2.pdf", fig2,
#  width=16.79,height=10.65,units="in",dpi=300)
```

Figure S1

In the online appendix we provide, for each locality, the number of Covid and non-Covid protests during our period of analysis. These descriptive statistics are presented in Figure S1.

```
# protest per type
df <- protest.event
df$protest_type <- as.factor(df$protest_type)
summary(df$protest_type)

# count protest events per type and locality
df2 <- df %>%
  group_by(locality_name, protest_type) %>%
  dplyr::summarise(count = length(locality_id))

df2$protest_type <-as.factor(df2$protest_type)

# drop NA locality_names
```

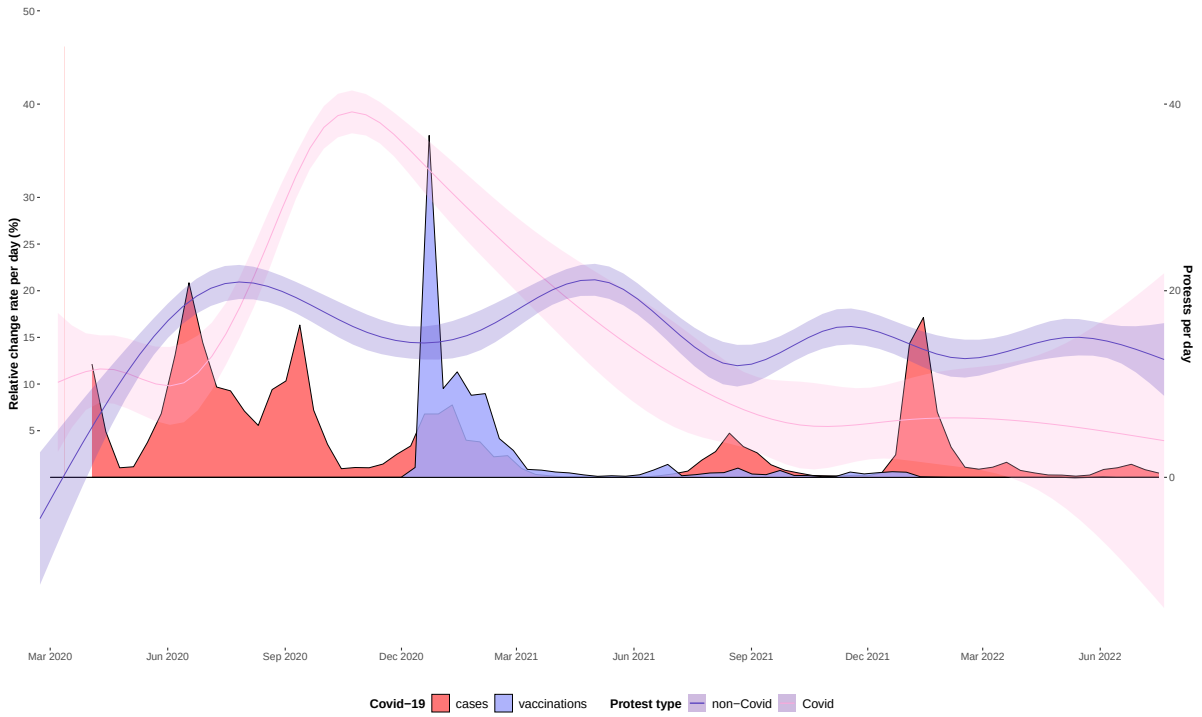

Figure 2: Number of daily Covid-19 (pink) and non Covid-19 protests (red) in Israeli municipalities relative to the daily change rate of Covid-19 cases (purple) and vaccinations (blue) between March 2020 and July 2022 (Figure 2).

```
df2 <- df2[!is.na(df2$locality_name),]

fig_s1 = ggplot() +
  geom_col(data = df2,
           aes(x = fct_rev(fct_reorder(locality_name, count)),
               y = count, fill = protest_type), alpha = .99,
           width = 0.9, position="stack",
           color= "black", linewidth= .1, na.rm = TRUE) +
  scale_fill_manual(labels = c("Covid","non-Covid"),
                    name = "Protest type" ,
                    values = c("#de2414","#ffb3de")) +
  scale_y_continuous(trans = "sqrt",
                     limits = c(0, 300),
                     breaks=c(5,10,25, 50, 150, 300),
                     labels=c("5","10","25","50","150","300")) +
  coord_flip() +
  labs(y = "protests", x = "local authorities") +
        theme(panel.grid.major = element_blank(),
           panel.grid.minor = element_blank(),
           panel.background = element_blank(),
           axis.text.x = element_text(size =10),
           axis.title.x = element_text(size =12, face = "bold"),
           axis.text.y =  element_text(size =8),
           axis.title.y = element_text(size =12, face = "bold"),
           legend.position="bottom", legend.box = "horizontal",
```

```
            legend.title = element_text(size=12, face="bold"),
            legend.text = element_text(size=12)) +
          labs(y="", x="")
fig_s1
```

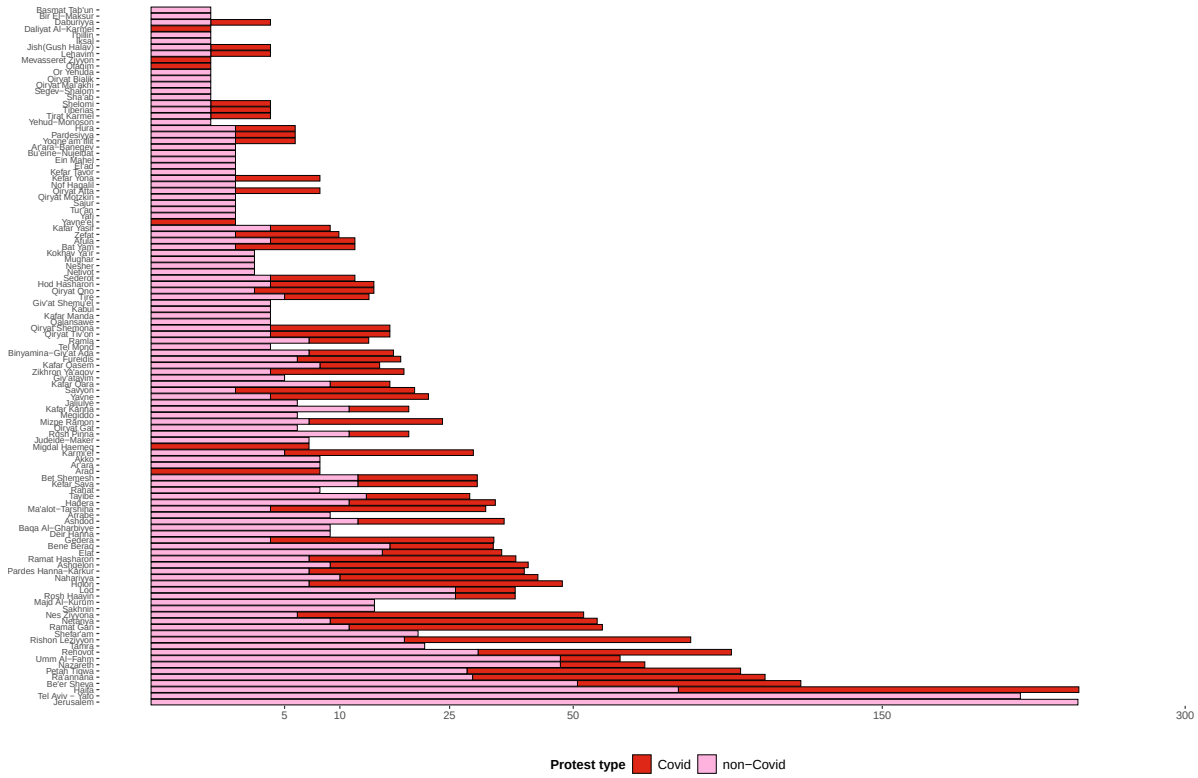

Figure 3: Protest counts per type and municipality (Figure S1).

```
# optional: save Figure S1 as .svg or .pdf
#ggsave("figures/si_fig_s1.svg",fig_s1,
#  width=16.79,height=25.65,units="in",dpi=300)
#ggsave("figures/si_fig_s1.pdf",fig_s1,
#  width=16.79,height=25.65,units="in",dpi=300)
```

2. Imputation and descriptive statistics

We first import all necessary helper packages and initialize the data. For easier accessibility, we also relabel all variables to have meaningful names that can be directly used in tables and figures.

```
# load helper functions and packages
source("scripts/helper_packages_load.R")
source("scripts/helper_functions.R")

# load packages
packages_load(install = FALSE)

# set seed
seed <- 42
set.seed(seed)
```

```r
# change variable names in main data frame used from here on forward
colnames(df_save)[6:65] <- c("Population density",
                  "Population size (log)", "Jewish population",
                  "Arab population",
                  "Jewish orthodox majority", "Muslim majority",
                  "Christian majority", "Druze majority",
                  "Residential housing planned",
                  "Residential housing constructed",
                  "Dependence ratio", "Birth rate", "Aged +65",
                  "Aged <18",
                  "Population growth rate", "Fertility rate",
                  "New inhabitants",
                  "New immigrants",
                  "Immigrant rate",  "Divorce rate", "Schools",
                  "Paid teacher rate",  "Children per teacher",
                  "High schools",  "High school students",
                  "Drop-out rate",
                  "University degree rate", "Student rate",
                  "Life expectancy",
                  "Diabetes rate",
                  "Overweight rate",
                  "Turnout local government", "Turnout Knesset 2019",
                  "Turnout Knesset 2020", "Turnout Knesset 2021",
                  "Municipality founding year",
                  "Municipality border beach",
                  "Municipality councillors",
                  "Municipality size",
                  "Municipality budget",
                  "Property tax (residential)" ,
                  "Property tax (commercial)",
                  "Property tax (industrial)",
                  "Property tax (total)",
                  "Unemployment rate",
                  "Unemployment mean age",
                  "Unemployment daily rate",
                  "Beneficiaries child support",
                  "Socio-economic index", "Socio-economic cluster",
                  "Municipality development",
                  "Distance to Tel Aviv",
                  "Peripherality index",
                  "Covid-19 cases",
                  "Covid-19 hospitalizations",
                  "Covid-19 tests",
                  "Covid-19 government stringency index",
                  "Covid-19 economic support index",
                  "Covid-19 containment health index",
                  "Covid-19 government response index")
```

There is a small number of missing information for the covariates we use, about 5.9 %, which we impute using k-nearest neighbor imputation. And we standardize all covariates prior to analysis. This code performs the imputation and standardization generating the information used in Table S1 in the online appendix that provides detailed descriptive statistics for each covariate dimension.

```r
# remove those columns (temporarily) for which we will not
# impute/standardize (week/locality ID, protest counts)
df <- df_save[,c(-1:-5)]
pro <- df_save[,c(3:5)]

# missing value rate
totalcells = prod(dim(df))
missingcells = sum(is.na(df))
percentage = (missingcells * 100 )/(totalcells)

percentage

# impute and standardize covariates
pre.model <- preProcess(df, method= c("knnImpute", "center", "scale"))
df <- predict(pre.model, newdata = df)
df <- data.frame(df)

# get protest counts back into the data frame
df <- cbind(pro, df)

# descriptive stats for categorical variables
prop.table(table(df$Jewish.orthodox.majority))
prop.table(table(df$Muslim.majority))
prop.table(table(df$Christian.majority))
prop.table(table(df$Druze.majority))

# descriptive stats for continuous variables
descr <- psych::describe(df, fast=TRUE)

# optional: export latex table with summary stats
#xtable(descr, out = "latex")

# update main data frame
df_save <- cbind(df_save[,1:2],df)

# pull out time column
week <- df_save$week_id

# pull out locality column for per locality partition
locality <- as.numeric(df_save$locality_id)
```

3. Generalized linear models

We first calculate a set of GLMs, separately for Covid and non-Covid protest and both for count and for binary outcomes (protest yes/no). We repeat this both for a spatial and temporal split. The count model with spatial split is our main model variant, all other models are reported in the online appendix.

3.1 Poisson regression for count outcomes

For the count outcome, we first use a spatial split to look at Covid protest (Section 3.1.1), then non-Covid protest outcomes (Section 3.1.2). And then we repeat this analysis with a temporal split for Covid (Section 3.1.3) and non-Covid protest outcomes (Section 3.1.4). Note that we calculate a full model and minimal model (with only the top-20 predictors) for each case.

3.1.1 Covid protest with spatial split

```
# subset to Covid protest
df <- df_save
df <- df[, c(-1:-3, -5)]

# rename protest variable
colnames(df)[1] = "protest"

# locality-based split (80/20)
# splits the data into training and testing sets, ensuring that all
# observations from a local government are grouped into the same subset
train.rows <- CreateLocalityPartition(df[, 1], locality = locality, p=0.8)
df.train <- df[train.rows,]
df.test <- df[-train.rows,]

# save df.test as .csv for cross-predictions in section 5
write.csv(df.test,"results/df.test.cross_glm_covid.csv", row.names = FALSE)

# pull out protest from training and test data
y_train <- df.train[, 1]
y_test <- df.test[, 1]

# define number of folds on protest
folds.10 <- createFolds(y_train, k = 10)

# define trainControl parameters
control <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 5,
  verboseIter = TRUE,
  savePredictions = "final",
  index = folds.10
)

# run count model from poisson family of glm
model_glm <- train(
  protest ~ .,
  data = df.train,
  method = "glm",
  family = "poisson",
  trControl = control
)
model_glm_covid.count_full <- model_glm

# optional: save best model as .Rda
#save(model_glm_covid.count_full,
#     file="results/model_glm_covid.count_full.Rda")

# in-sample prediction: model fit
glm.fitted <- predict(model_glm)

# out-of-sample prediction: model performance
```

```r
glm.predicted <- predict(model_glm, df.test)

# calculate performance statistics ("in" and "out" of sample)
performance <- list(
  RMSE.in = rmse(df.train$protest, glm.fitted),
  cor.in = cor(df.train$protest, glm.fitted),
  R2.in = cor(df.train$protest, glm.fitted)^2,
  MAE.in = mae(df.train$protest, glm.fitted),
  RMSE.out = rmse(df.test$protest, glm.predicted),
  cor.out = cor(df.test$protest, glm.predicted),
  R2.out = cor(df.test$protest, glm.predicted)^2,
  MAE.out = mae(df.test$protest, glm.predicted))

# save performance statistics
performance_glm_covid.count_full <- performance

# calculate variable importance
drivers <- varImp(model_glm$finalModel, value = "gcv")

# export most important variables
varImp <- data.frame(value = drivers$Overall, names = rownames(drivers))
varImp <- varImp[order(varImp$value, decreasing = TRUE),]

# save top-20 variables
top20 <- varImp[c(1:20),]
top20.vec <- top20$names

# minimal model

# subset train and test data to top 20 variables identified by full model
df.train <- df.train[,names(df.train) %in% top20.vec]
df.test <- df.test[,names(df.test) %in% top20.vec]

# add protest to training and test data
df.train$protest <- y_train
df.test$protest <- y_test

# run count model from poisson family of glm
model_glm <- train(
  protest ~ .,
  data = df.train,
  method = "glm",
  family = "poisson",
  trControl = control
)
model_glm_covid.count_min <- model_glm

# save best model as .Rda for cross-prediction in section 5
save(model_glm_covid.count_min,
     file="results/model_glm_covid.count_min.Rda")

# get and save final model (used in section 3.3)
final.model_glm_covid.count_min <- model_glm$finalModel
```

```r
# in-sample prediction: model fit
glm.fitted <- predict(model_glm)

# out-of-sample prediction: model performance
glm.predicted <- predict(model_glm, df.test)

# calculate performance statistics ("in" and "out" of sample)
performance <- list(
  RMSE.in = rmse(df.train$protest, glm.fitted),
  cor.in = cor(df.train$protest, glm.fitted),
  R2.in = cor(df.train$protest, glm.fitted)^2,
  MAE.in = mae(df.train$protest, glm.fitted),
  RMSE.out = rmse(df.test$protest, glm.predicted),
  cor.out = cor(df.test$protest, glm.predicted),
  R2.out = cor(df.test$protest, glm.predicted)^2,
  MAE.out = mae(df.test$protest, glm.predicted))

# save performance statistics
performance_glm_covid.count_min <- performance

# calculate variable importance
drivers <- varImp(model_glm$finalModel, value = "gcv")

# export most important variables
varImp <- data.frame(value = drivers$Overall, names = rownames(drivers))
varImp <- varImp[order(varImp$value, decreasing = TRUE),]

# save and export top-20 variables
top20 <- varImp[c(1:20),]
top20.glm_covid.count_min <- varImp[c(1:20),]
```

3.1.2 Non-Covid protest with spatial split

```r
# subset to non-Covid protest
df <- df_save
df <- df[, c(-1:-4)]

# rename protest variable
colnames(df)[1] = "protest"

# locality-based split (80/20)
# splits the data into training and testing sets, ensuring that all
# observations from a local government are grouped into the same subset
train.rows <- CreateLocalityPartition(df[, 1], locality = locality, p=0.8)
df.train <- df[train.rows,]
df.test <- df[-train.rows,]

# save df.test as .csv for cross-predictions in section 5
write.csv(df.test,
          "results/df.test.cross_glm_non_covid.csv",row.names = FALSE)

# pull out protest from training and test data
y_train <- df.train[, 1]
```

```r
y_test <- df.test[, 1]

# define number of folds on protest
folds.10 <- createFolds(y_train, k = 10)

# define trainControl parameters
control <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 5,
  verboseIter = TRUE,
  savePredictions = "final",
  index = folds.10
)

# run count model from poisson family of glm
model_glm <- train(
  protest ~ .,
  data = df.train,
  method = "glm",
  family = "poisson",
  trControl = control
)
model_glm_non_covid.count_full <- model_glm

# optional: save best model as .Rda
#save(model_glm_non_covid.count_full,
#file="results/model_glm_non_covid.count_full.Rda")

# in-sample prediction: model fit
glm.fitted <- predict(model_glm)

# out-of-sample prediction: model performance
glm.predicted <- predict(model_glm, df.test)

# calculate performance statistics ("in" and "out" of sample)
performance <- list(
  RMSE.in = rmse(df.train$protest, glm.fitted),
  cor.in = cor(df.train$protest, glm.fitted),
  R2.in = cor(df.train$protest, glm.fitted)^2,
  MAE.in = mae(df.train$protest, glm.fitted),
  RMSE.out = rmse(df.test$protest, glm.predicted),
  cor.out = cor(df.test$protest, glm.predicted),
  R2.out = cor(df.test$protest, glm.predicted)^2,
  MAE.out = mae(df.test$protest, glm.predicted))

# save performance statistics
performance_glm_non_covid.count_full <- performance

# calculate variable importance
drivers <- varImp(model_glm$finalModel, value = "gcv")

# export most important variables
```

```r
varImp <- data.frame(value = drivers$Overall, names = rownames(drivers))
varImp <- varImp[order(varImp$value, decreasing = TRUE),]

# save top-20 variables
top20 <- varImp[c(1:20),]
top20.vec <- top20$names

# minimal model

# subset train and test data to top-20 identified by full model
df.train <- df.train[,names(df.train) %in% top20.vec]
df.test <- df.test[,names(df.test) %in% top20.vec]

# add protest to training and test data
df.train$protest <- y_train
df.test$protest <- y_test

# run count model from poisson family of glm
model_glm <- train(
  protest ~ .,
  data = df.train,
  method = "glm",
  family = "poisson",
  trControl = control
)
model_glm_non_covid.count_min <- model_glm

# save best model as .Rda for cross-prediction in section 5
save(model_glm_non_covid.count_min,
     file="results/model_glm_non_covid.count_min.Rda")

# get and save final model (used in section 3.3)
final.model_glm_non_covid.count_min <- model_glm$finalModel

# in-sample prediction: model fit
glm.fitted <- predict(model_glm)

# out-of-sample prediction: model performance
glm.predicted <- predict(model_glm, df.test)

# calculate performance statistics ("in" and "out" of sample)
performance <- list(
  RMSE.in = rmse(df.train$protest, glm.fitted),
  cor.in = cor(df.train$protest, glm.fitted),
  R2.in = cor(df.train$protest, glm.fitted)^2,
  MAE.in = mae(df.train$protest, glm.fitted),
  RMSE.out = rmse(df.test$protest, glm.predicted),
  cor.out = cor(df.test$protest, glm.predicted),
  R2.out = cor(df.test$protest, glm.predicted)^2,
  MAE.out = mae(df.test$protest, glm.predicted))

# save performance statistics
performance_glm_non_covid.count_min <- performance
```

```r
# calculate variable importance
drivers <- varImp(model_glm$finalModel, value = "gcv")

# export most important variables
varImp <- data.frame(value = drivers$Overall, names = rownames(drivers))
varImp <- varImp[order(varImp$value, decreasing = TRUE),]

# save top-20 variables
top20 <- varImp[c(1:20),]
top20.glm_non_covid.count_min <- varImp[c(1:20),]
```

3.1.3 Covid protest with temporal split

```r
# subset to Covid protest
df <- df_save
df <- df[, c(-1:-3, -5)]

# rename protest variable
colnames(df)[1] = "protest"

# temporal split sample
# splits the data after 95 weeks into training and test set
df.train <- df[week <= 95,]
df.test <- df[week > 95,]

# pull out protest from training and test data
y_train <- df.train[, 1]
y_test <- df.test[, 1]

# define number of folds on protest
folds.10 <- createFolds(y_train, k = 10)

# define trainControl parameters
control <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 5,
  verboseIter = TRUE,
  savePredictions = "final",
  index = folds.10
)

# run count model from poisson family of glm
model_glm <- train(
  protest ~ .,
  data = df.train,
  method = "glm",
  family = "poisson",
  trControl = control
)
model_glm_covid.count.temp_full <- model_glm

# optional: save best model as .Rda
```

```r
#save(model_glm_covid.count.temp_full,
#     file="results/model_glm_covid.count.temp_full.Rda")

# in-sample prediction: model fit
glm.fitted <- predict(model_glm)

# out-of-sample prediction: model performance
glm.predicted <- predict(model_glm, df.test)

# calculate performance statistics ("in" and "out" of sample)
performance <- list(
  RMSE.in = rmse(df.train$protest, glm.fitted),
  cor.in = cor(df.train$protest, glm.fitted),
  R2.in = cor(df.train$protest, glm.fitted)^2,
  MAE.in = mae(df.train$protest, glm.fitted),
  RMSE.out = rmse(df.test$protest, glm.predicted),
  cor.out = cor(df.test$protest, glm.predicted),
  R2.out = cor(df.test$protest, glm.predicted)^2,
  MAE.out = mae(df.test$protest, glm.predicted))

# save performance statistics
performance_glm_covid.count.temp_full <- performance

# calculate variable importance
drivers <- varImp(model_glm$finalModel, value = "gcv")

# export most important variables
varImp <- data.frame(value = drivers$Overall, names = rownames(drivers))
varImp <- varImp[order(varImp$value, decreasing = TRUE),]

# save top-20 variables
top20 <- varImp[c(1:20),]
top20.vec <- top20$names

# minimal model

# subset train and test data to top 20 identified by full model
df.train <- df.train[,names(df.train) %in% top20.vec]
df.test <- df.test[,names(df.test) %in% top20.vec]

# add protest to training and test data
df.train$protest <- y_train
df.test$protest <- y_test

# run count model from poisson family of glm
model_glm <- train(
  protest ~ .,
  data = df.train,
  method = "glm",
  family = "poisson",
  trControl = control
)
model_glm_covid.count.temp_min <- model_glm
```

```r
# optional: save best model as .Rda
#save(model_glm_covid.count.temp_min,
#     file="results/model_glm_covid.count.temp_min.Rda")

# get and save final model (used in section 3.3)
final.model_glm_covid.count.temp_min <- model_glm$finalModel

# in-sample prediction: model fit
glm.fitted <- predict(model_glm)

# out-of-sample prediction: model performance
glm.predicted <- predict(model_glm, df.test)

# calculate performance statistics ("in" and "out" of sample)
performance <- list(
  RMSE.in = rmse(df.train$protest, glm.fitted),
  cor.in = cor(df.train$protest, glm.fitted),
  R2.in = cor(df.train$protest, glm.fitted)^2,
  MAE.in = mae(df.train$protest, glm.fitted),
  RMSE.out = rmse(df.test$protest, glm.predicted),
  cor.out = cor(df.test$protest, glm.predicted),
  R2.out = cor(df.test$protest, glm.predicted)^2,
  MAE.out = mae(df.test$protest, glm.predicted))

# save performance statistics
performance_glm_covid.count.temp_min <- performance

# calculate variable importance
drivers <- varImp(model_glm$finalModel, value = "gcv")

# export most important variables
varImp <- data.frame(value = drivers$Overall, names = rownames(drivers))
varImp <- varImp[order(varImp$value, decreasing = TRUE),]

# save top-20 variables
top20 <- varImp[c(1:20),]
```

3.1.4 Non-Covid protest with temporal split

```r
# subset to non-Covid protest
df <- df_save
df <- df[, c(-1:-4)]

# rename protest variable
colnames(df)[1] = "protest"

# temporal split sample
# splits the data after 95 weeks into training and test set
df.train <- df[week <= 95,]
df.test <- df[week > 95,]

# pull out protest from training and test data
y_train <- df.train[, 1]
```

```r
y_test <- df.test[, 1]

# define number of folds on protest
folds.10 <- createFolds(y_train, k = 10)

# define trainControl parameters
control <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 5,
  verboseIter = TRUE,
  savePredictions = "final",
  index = folds.10
)

# run count model from poisson family of glm
model_glm <- train(
  protest ~ .,
  data = df.train,
  method = "glm",
  family = "poisson",
  trControl = control
)
model_glm_non_covid.count.temp_full <- model_glm

# optional: save best model as .Rda
#save(model_glm_non_covid.count.temp_full,
#     file="results/model_glm_non_covid.count.temp_full.Rda")

# in-sample prediction: model fit
glm.fitted <- predict(model_glm)

# out-of-sample prediction: model performance
glm.predicted <- predict(model_glm, df.test)

# calculate performance statistics ("in" and "out" of sample)
performance <- list(
  RMSE.in = rmse(df.train$protest, glm.fitted),
  cor.in = cor(df.train$protest, glm.fitted),
  R2.in = cor(df.train$protest, glm.fitted)^2,
  MAE.in = mae(df.train$protest, glm.fitted),
  RMSE.out = rmse(df.test$protest, glm.predicted),
  cor.out = cor(df.test$protest, glm.predicted),
  R2.out = cor(df.test$protest, glm.predicted)^2,
  MAE.out = mae(df.test$protest, glm.predicted))

# save performance statistics
performance_glm_non_covid.count.temp_full <- performance

# calculate variable importance
drivers <- varImp(model_glm$finalModel, value = "gcv")

# export most important variables
```

```r
varImp <- data.frame(value = drivers$Overall, names = rownames(drivers))
varImp <- varImp[order(varImp$value, decreasing = TRUE),]

# save top-20 variables
top20 <- varImp[c(1:20),]
top20.vec <- top20$names

# minimal model

# subset train and test data to top 20 identified by full model
df.train <- df.train[,names(df.train) %in% top20.vec]
df.test <- df.test[,names(df.test) %in% top20.vec]

# add protest to training and test data
df.train$protest <- y_train
df.test$protest <- y_test

# run count model from poisson family of glm
model_glm <- train(
  protest ~ .,
  data = df.train,
  method = "glm",
  family = "poisson",
  trControl = control
)
model_glm_non_covid.count.temp_min <- model_glm

# optional: save best model as .Rda
#save(model_glm_non_covid.count.temp_min,
#     file="results/model_glm_non_covid.count.temp_min.Rda")

# get and save final model (used in section 3.3)
final.model_glm_non_covid.count.temp_min<-model_glm$finalModel

# in-sample prediction: model fit
glm.fitted <- predict(model_glm)

# out-of-sample prediction: model performance
glm.predicted <- predict(model_glm, df.test)

# calculate performance statistics ("in" and "out" of sample)
performance <- list(
  RMSE.in = rmse(df.train$protest, glm.fitted),
  cor.in = cor(df.train$protest, glm.fitted),
  R2.in = cor(df.train$protest, glm.fitted)^2,
  MAE.in = mae(df.train$protest, glm.fitted),
  RMSE.out = rmse(df.test$protest, glm.predicted),
  cor.out = cor(df.test$protest, glm.predicted),
  R2.out = cor(df.test$protest, glm.predicted)^2,
  MAE.out = mae(df.test$protest, glm.predicted))

# save performance statistics
performance_glm_non_covid.count.temp_min <- performance
```

```r
# calculate variable importance
drivers <- varImp(model_glm$finalModel, value = "gcv")

# export most important variables
varImp <- data.frame(value = drivers$Overall, names = rownames(drivers))
varImp <- varImp[order(varImp$value, decreasing = TRUE),]

# save top-20 variables
top20 <- varImp[c(1:20),]
```

3.2 Binomial regression for binary outcomes

Again, we use a spatial split to look at Covid protest (Section 3.2.1), then non-Covid protest outcomes (Section 3.2.2), this time using a binomial regression model for binary protest outcomes (yes/no). Note that we also calculate a full model and minimal model (with only the top-20 predictors) for each case.

3.2.1 Covid protest with spatial split

```r
# subset to Covid protest
df <- df_save
df <- df[, c(-1:-3, -5)]

# rename protest variable
colnames(df)[1] = "protest"

# code binary outcome variable
df$protest <- as.factor(ifelse(df$protest  >= 1, "1", "0"))

# locality-based split (80/20)
# splits the data into training and testing sets, ensuring that all
# observations from a local government are grouped into the same subset
train.rows <- CreateLocalityPartition(df[, 1], locality = locality, p=0.8)
df.train <- df[train.rows,]
df.test <- df[-train.rows,]

# pull out protest from training and test data
y_train <- df.train[, 1]
y_test <- df.test[, 1]

# define number of folds on protest
folds.10 <- createFolds(y_train, k = 10)

# define parameters
control <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 5,
  verboseIter = TRUE,
  savePredictions = "final",
  search="grid",
  index = folds.10
)
```

```r
# run binomial regression from family of glm
model_glm <- train(
  protest ~ .,
  data = df.train,
  method = "glm",
  family = "binomial",
  metric = "Accuracy",
  trControl = control
)
model_glm_covid.bin_full <- model_glm

# optional: save best model as .Rda
#save(model_glm_covid.bin_full,
#     file="results/model_glm_covid.bin_full.Rda")

# in-sample prediction: model fit
glm.fitted <- predict(model_glm)

# out-of-sample prediction: model performance
glm.predicted <- predict(model_glm, df.test)

# confusion matrix: in-sample
cm.in <- confusionMatrix(reference = df.train$protest,
                         data = glm.fitted,
                         positive = "1",
                         mode = "prec_recall")

# confusion matrix: out-of-sample
cm.out <- confusionMatrix(reference = df.test$protest,
                         data = glm.predicted,
                         positive = "1",
                         mode = "prec_recall")

# save performance statistics ("in" and "out" of sample)
performance.in_glm_covid.bin_full <- cm.in
performance.out_glm_covid.bin_full <- cm.out

# calculate variable importance
drivers <- varImp(model_glm$finalModel, value = "gcv")

# export most important variables
varImp <- data.frame(value = drivers$Overall, names=rownames(drivers))
varImp <- varImp[order(varImp$value, decreasing = TRUE),]

# save top-20 variables
top20 <- varImp[c(1:20),]
top20.vec <- top20$names

# minimal model

# subset train and test data to top 20 identified by full model
df.train <- df.train[,names(df.train) %in% top20.vec]
df.test <- df.test[,names(df.test) %in% top20.vec]
```

```r
# add protest to training and test data
df.train$protest <- y_train
df.test$protest <- y_test

# run binomial regression from family of glm
model_glm <- train(
  protest ~ .,
  data = df.train,
  method = "glm",
  family = "binomial",
  metric = "Accuracy",
  trControl = control
)
model_glm_covid.bin_min <- model_glm

# optional: save best model as .Rda
#save(model_glm_covid.bin_min,
#     file="results/model_glm_covid.bin_min.Rda")

# get and save final model (used in section 3.3)
final.model_glm_covid.bin_min <- model_glm$finalModel

# in-sample prediction: model fit
glm.fitted <- predict(model_glm)

# out-of-sample prediction: model performance
glm.predicted <- predict(model_glm, df.test)

# confusion matrix: in-sample
cm.in <- confusionMatrix(reference = df.train$protest,
                         data = glm.fitted,
                         positive = "1",
                         mode = "prec_recall")

# confusion matrix: out-of-sample
cm.out <- confusionMatrix(reference = df.test$protest,
                         data = glm.predicted,
                         positive = "1",
                         mode = "prec_recall")

# save performance statistics ("in" and "out" of sample)
performance.in_glm_covid.bin_min <- cm.in
performance.out_glm_covid.bin_min <- cm.out

# calculate variable importance
drivers <- varImp(model_glm$finalModel, value = "gcv")

# export most important variables
varImp <- data.frame(value = drivers$Overall, names = rownames(drivers))
varImp <- varImp[order(varImp$value, decreasing = TRUE),]

# save top-20 variables
top20 <- varImp[c(1:20),]
```

3.2.2 Non-Covid protest with spatial split

```
# subset to non-Covid protest
df <- df_save
df <- df[, c(-1:-4)]

# rename protest variable
colnames(df)[1] = "protest"

# code binary outcome variable
df$protest <- as.factor(ifelse(df$protest  >= 1,"1","0"))

# locality-based split (80/20)
# splits the data into training and testing sets, ensuring that all
# observations from a local government are grouped into the same subset
train.rows <- CreateLocalityPartition(df[, 1], locality = locality, p=0.8)
df.train <- df[train.rows,]
df.test <- df[-train.rows,]

# pull out protest from training and test data
y_train <- df.train[, 1]
y_test <- df.test[, 1]

# define number of folds on protest
folds.10 <- createFolds(y_train, k = 10)

# define parameters
control <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 5,
  verboseIter = TRUE,
  savePredictions = "final",
  search="grid",
  index = folds.10
)

# run binomial regression from family of glm
model_glm <- train(
  protest ~ .,
  data = df.train,
  method = "glm",
  family = "binomial",
  metric = "Accuracy",
  trControl = control
)
model_glm_non_covid.bin_full <- model_glm

# optional: save best model as .Rda
#save(model_glm_non_covid.bin_full,
#     file="results/model_glm_non_covid.bin_full.Rda")

# get and save final model (used in section 3.3)
final.model_glm_non_covid.bin_min <- model_glm$finalModel
```

```r
# in-sample prediction: model fit
glm.fitted <- predict(model_glm)

# out-of-sample prediction: model performance
glm.predicted <- predict(model_glm, df.test)

# confusion matrix: in-sample
cm.in <- confusionMatrix(reference = df.train$protest,
                         data = glm.fitted,
                         positive = "1",
                         mode = "prec_recall")

# confusion matrix: out-of-sample
cm.out <- confusionMatrix(reference = df.test$protest,
                         data = glm.predicted,
                         positive = "1",
                         mode = "prec_recall")

# save performance statistics ("in" and "out" of sample)
performance.in_glm_non_covid.bin_full <- cm.in
performance.out_glm_non_covid.bin_full <- cm.out

# calculate variable importance
drivers <- varImp(model_glm$finalModel, value = "gcv")

# export most important variables
varImp <- data.frame(value = drivers$Overall, names = rownames(drivers))
varImp <- varImp[order(varImp$value, decreasing = TRUE),]

# save top-20 variables
top20 <- varImp[c(1:20),]
top20.vec <- top20$names

# minimal model

# subset train and test data to top 20 identified by full model
df.train <- df.train[,names(df.train) %in% top20.vec]
df.test <- df.test[,names(df.test) %in% top20.vec]

# add protest to training and test data
df.train$protest <- y_train
df.test$protest <- y_test

# run binomial regression from family of glm
model_glm <- train(
  protest ~ .,
  data = df.train,
  method = "glm",
  family = "binomial",
  metric = "Accuracy",
  trControl = control
)
model_glm_non_covid.bin_min <- model_glm
```

```r
# optional: save best model as .Rda
#save(model_glm_non_covid.bin_min,
#     file="results/model_glm_non_covid.bin_min.Rda")

# in-sample prediction: model fit
glm.fitted <- predict(model_glm)

# out-of-sample prediction: model performance
glm.predicted <- predict(model_glm, df.test)

# confusion matrix: in-sample
cm.in <- confusionMatrix(reference = df.train$protest,
                         data = glm.fitted,
                         positive = "1",
                         mode = "prec_recall")

# confusion matrix: out-of-sample
cm.out <- confusionMatrix(reference = df.test$protest,
                         data = glm.predicted,
                         positive = "1",
                         mode = "prec_recall")

# save performance statistics ("in" and "out" of sample)
performance.in_glm_non_covid.bin_min <- cm.in
performance.out_glm_non_covid.bin_min <- cm.out

# calculate variable importance
drivers <- varImp(model_glm$finalModel, value = "gcv")

# export most important variables
varImp <- data.frame(value = drivers$Overall, names = rownames(drivers))
varImp <- varImp[order(varImp$value, decreasing = TRUE),]

# save top-20 variables
top20 <- varImp[c(1:20),]
```

3.3 Model Outputs

The following code provides model results for all GLM sub-models in latex, corresponding to Tables S5, S6 and S08 in the online appendix.

```r
# GLM: minimal poisson regression model (spatial split)
texreg(list(final.model_glm_covid.count_min,
            final.model_glm_non_covid.count_min),
       booktabs = TRUE,
       stars = c(0.01,0.05, 0.001),
       custom.model.names = c("Covid protest", "Non-Covid protest"),
       caption = "Minimal Poisson regression results.",
       label = "stab:05",
       type = "latex",
       float.pos = "h",
       single.row = TRUE,
       robust = TRUE,
```

```
       include.adjrs = FALSE,
       dcolumn = FALSE,
       include.bic = FALSE)

# GLM: minimal poisson regression model (temporal split)
texreg(list(final.model_glm_covid.count.temp_min,
            final.model_glm_non_covid.count.temp_min),
       booktabs = TRUE,
       stars = c(0.01,0.05, 0.001),
       custom.model.names = c("Covid protest", "Non-Covid protest"),
       caption = "Temporal split sample: Minimal Poisson regression results.",
       label = "stab:06",
       type = "latex",
       float.pos = "h",
       single.row = TRUE,
       robust = TRUE,
       include.adjrs = FALSE,
       dcolumn = FALSE,
       include.bic = FALSE)

# GLM: minimal binomial regression model (spatial split)
texreg(list(final.model_glm_covid.bin_min,
            final.model_glm_non_covid.bin_min),
       booktabs = TRUE,
       stars = c(0.01,0.05, 0.001),
       custom.model.names = c("Covid protest", "Non-Covid protest"),
       caption = "Minimal Binomial regression results.",
       label = "stab:08",
       type = "latex",
       float.pos = "h",
       single.row = TRUE,
       robust = TRUE,
       include.adjrs = FALSE,
       dcolumn = FALSE,
       include.bic = FALSE)
```

Figure S2

Figure S2 illustrates the top-20 important drivers of protest, per type and based on the minimal Poisson regression model.

```
# get (sorted) top-20 Covid protest drivers
drivers.cov <- top20.glm_covid.count_min

# get (sorted) top-20 non-Covid protest drivers
drivers.noncov <- top20.glm_non_covid.count_min

# convert to relative importance values (in %)
drivers.cov.total <- sum(drivers.cov$value)
drivers.cov$value <- -100*drivers.cov$value/drivers.cov.total
drivers.noncov[is.na(drivers.noncov$value),]$value <- 0
drivers.noncov.total <- sum(drivers.noncov$value)
drivers.noncov$value <- 100*drivers.noncov$value/drivers.noncov.total
```

```r
# add missing rows from one df in the other
drivers <- merge(drivers.cov, drivers.noncov, by= c("names"),
                 all.x = TRUE, all.y = TRUE)

drivers.covid <- drivers[, c(1,2)]
colnames(drivers.covid)[2] ="value"
drivers.covid

drivers.noncov <- drivers[, c(1,3)]
colnames(drivers.noncov)[2] ="value"
drivers.noncov

# add column with protest type
drivers.covid$type <- "Covid protests"
drivers.noncov$type <- "Non-Covid protests"

# replace NA with 0
drivers.covid[is.na(drivers.covid$value),]$value <- 0
drivers.noncov[is.na(drivers.noncov$value),]$value <- 0

# rbind
df.plot <- rbind(drivers.covid, drivers.noncov)

# import list with variables and mechanisms
df.mechanism <- read.csv("data/raw_mechanism.csv")

# get mechanisms and variable names
df.plot <- merge(df.plot, df.mechanism, by="names", all.x=TRUE)
df.plot <- df.plot[!is.na(df.plot$indicator),]
df.plot$type <- as.factor(df.plot$type)

# indicator levels
df.plot <- df.plot[order(df.plot$type, df.plot$value),]
lvls <- df.plot$indicator
lvls <- unique(lvls)
df.plot$indicator <- factor(df.plot$indicator, levels=c(lvls))

# mechanism levels
lvls.m <- c("Opportunity", "Coercion", "Structure", "Grievance")
df.plot$mechanism <- factor(df.plot$mechanism, levels=c(lvls.m))

##### plot
fig_s02 <- ggplot2::ggplot() +
  geom_bar(data = df.plot,
           aes(x=value, y=fct_rev(indicator), fill= mechanism),
           stat = "identity", colour="black", linewidth= .1,
           width = .75, alpha = 0.9, na.rm = TRUE) +
  scale_fill_manual(name = "",
                    values = c("#de2414","#ff6161",
                               "#ff6699","#ffb3de"),
                    labels = c("Opportunity", "Coercion",
                               "Structure","Grievance"),
                    na.translate = F) +
```

```
  annotate("text", x = c(-6, 6), y = c(34.15,34.15),
           label = c("Covid protest", "Non-Covid protest"),
           size = 4.5, fontface = "bold") +
  ggplot2::labs(y = " ",
                x = "GINI Variable importance (scaled)") +
  scale_y_discrete(expand=c(0.1,0)) +
  scale_x_continuous(limits = c(-12.5, 12.5),
                     breaks=c(-12,-10,-8,-6,-4,-2,0,
                              2,4,6,8,10,12),
                     labels=c("12","10","8","6","4","2","0",
                              "2","4","6","8","10","12")) +
  theme_minimal() +
  theme(panel.grid.major.x = element_blank(),
        panel.grid.minor.x = element_blank(),
        panel.grid.minor.y = element_blank(),
        panel.grid.major.y = element_blank(),
        axis.text.x = element_text(size =10),
        axis.title.x  = element_text(size = 12, face = "bold"),
        axis.text.y =  element_text(size =10),
        axis.title.y = element_text(size=12),
        legend.position= c(.5, 1.04),
        legend.direction = "horizontal",
        legend.text = element_text(size=12),
        plot.title = element_blank(),
        plot.subtitle = element_blank(),
        plot.background = element_blank(),
        plot.margin=unit(c(4, 0.5, 2, 0.5), units="line")) #t, r, b, l)

fig_s02
```

```
# optional: save Figure S02 as .svg or .pdf
#ggsave("figures/si_fig_s02.svg", fig_s02,
#       width=16.79, height=10.65, units="in", dpi=300)
#ggsave("figures/si_fig_s02.pdf", fig_s02,
#       width=16.79,height=10.65, units="in", dpi=300)
```

4. Random forest models

We then use random forests for the same set of sub-models, i.e. separately for Covid and non-Covid protest and both for count and for binary outcomes (protest yes/no). We also repeat the analysis for a spatial and temporal split sample, and include cross-predictions. Note that we identify a full model and minimal model (with only the top-20 predictors) for each case based on cross-validation. We fine-tune the models using a custom grid search logic with no maximum number of nodes and different ntree (number of branches that grow after each split) and mytry (number of variables randomly collected to be sampled at each split), and select the most parsimonious specification with the smallest RMSE. Again, the count model with spatial split is our main model variant, all other models are reported in the online appendix.

Please note that the computing time for the random forest model can be several hours (depending on the hardware even longer). To minimize computing time for the replication package, we include the full code below but do not execute it. We provide results for each random forest sub-model as R-data frames (.RData) the sub-directory `/results`. The sub-directory `/rf_optimization` features cross-validation results. Each analysis can, of course, be run by changing the execution options below to fully replicate each model.

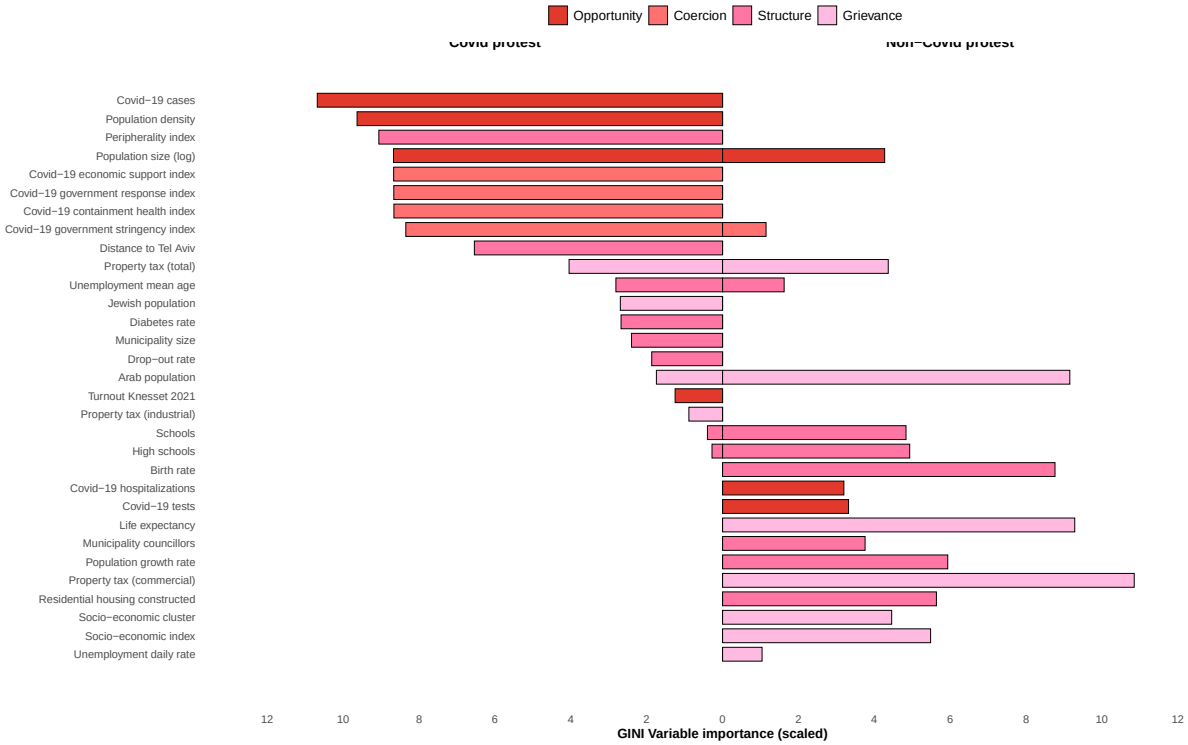

Figure 4: Top-20 important drivers of protest, per type and based on the minimal Poisson regression model (Figure S2).

4.1 Random forest count model

For the count outcome, we first use a spatial split to look at Covid protest (Section 4.1.1), then non-Covid protest outcomes (Section 4.1.2). And then we repeat this analysis with a temporal split for Covid (Section 4.1.3) and non-Covid protest outcomes (Section 4.1.4).

4.1.1 Covid protest with spatial split

```
# subset to Covid protest
df <- df_save
df <- df[, c(-1:-3, -5)]

# rename protest variable
colnames(df)[1] = "protest"

# one-hot encoding of categorical variables
oneHot.model <- dummyVars(" ~ .", data=df)
df <- predict(oneHot.model, newdata = df)
df <- data.frame(df)

# locality-based split (80/20)
# splits the data into training and testing sets, ensuring that all
# observations from a local government are grouped into the same subset
train.rows <- CreateLocalityPartition(df[, 1], locality = locality, p=0.8)
df.train <- df[train.rows,]
df.test <- df[-train.rows,]
```

```r
# save df.test as .csv for cross-prediction in section 5
write.csv(df.test,"results/df.test.cross_rf_covid.csv", row.names = FALSE)

# pull out protest from training and test data
y_train <- df.train[, 1]
y_test <- df.test[, 1]

# define number of folds on protest
folds.10 <- createFolds(y_train, k = 10)

# specify 10 folds and repeat 5 times
control <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 5,
  verboseIter = TRUE,
  savePredictions = "final",
  search="grid",
  index = folds.10)

# create tune grid with values from 3:12 for mtry
tunegrid <- expand.grid(.mtry=c(3:12))

# define ntrees, i.e. the number of trees to grow
ntrees <- c(50,80,100,200,500,1000,1500,2000)

# run grid search
modellist <- list()
for (ntree in ntrees) {
  fit = train(
    protest ~ .,
    tuneLength = 5,
    importance = TRUE,
    data = df.train,
    method = "rf",
    trControl = control,
    tuneGrid = tunegrid,
    ntree = ntree)
  key <- toString(ntree)
  modellist[[key]] <- fit
}
# optional: save scan configuration
#save(modellist,file="rf_optimization/covid.count_full_modellist.Rda")

# compare results
results <- resamples(modellist)
summary(results)
pdf(file = "figures/rf_covid.count_full.pdf", width = 6, height = 3.5)
dotplot(results)
dev.off()

# select best model based on smallest RMSE/MAE and greatest Rsquared
ntree_opt <- "1000"
```

```r
model_rf <- modellist[[ntree_opt]]
mtry_opt <- as.numeric(unlist(model_rf$bestTune))

# optional: save best model as .Rda
#model_rf_covid.count_full <- modellist[[ntree_opt]]
#save(model_rf_covid.count_full,
#     file="results/model_rf_covid.count_full.Rda")

# in-sample prediction: model fit
rf.fitted <- predict(model_rf)

# out-of-sample prediction: model performance
rf.predicted <- predict(model_rf, df.test)

# calculate performance statistics ("in" and "out" of sample)
performance <- list(
  RMSE.in = rmse(df.train$protest, rf.fitted),
  cor.in = cor(df.train$protest, rf.fitted),
  R2.in = cor(df.train$protest, rf.fitted)^2,
  MAE.in = mae(df.train$protest, rf.fitted),
  RMSE.out = rmse(df.test$protest, rf.predicted),
  cor.out = cor(df.test$protest, rf.predicted),
  R2.out = cor(df.test$protest, rf.predicted)^2,
  MAE.out = mae(df.test$protest, rf.predicted))

# save performance statistics
performance_rf_covid.count_full <- performance
save(performance_rf_covid.count_full,
     file = "results/performance_rf_covid.count_full.Rda")

# variable permutation importance

# rerun classification using cforest from party pkg
rf.corr <- party::cforest(
  protest ~ .,
  data = df.train,
  control = cforest_unbiased(mtry = mtry_opt, ntree = as.numeric(ntree_opt))
)

# calculate unconditional variable importance
# (using faster implementation of permimp pkg)
varImp <- permimp::permimp(rf.corr, conditional = FALSE, asParty = TRUE)
varImp <- sort(varImp, decreasing = TRUE)

# get top-20 variables
top20 <- data.frame(varImp[c(1:20)])
top20$names <- row.names(top20)
row.names(top20) <- NULL
names(top20) <- c("%IncMSE", "names")
top20.vec <- top20$names

# save top-20 variables as .csv (used in section 4.3)
write.csv(top20,"results/top20.rf_covid.count_full.csv",
```

```
          row.names = FALSE)

# optional:
# calculate conditional variable importance
# (using faster implementation of permimp pkg)
#.          [Note: Very long compute times!]
#varImp_cond <- permimp::permimp(rf.corr, conditional = TRUE)
#varImp_cond <- sort(varImp_cond$values, decreasing = TRUE)
#save(varImp_cond, file="results/varImp_cond.rf_covid.count_full.Rda")

# optional: get top-20 variables
#top20_cond <- data.frame(varImp_cond[c(1:20)])
#top20_cond$names <- row.names(top20_cond)
#row.names(top20_cond) <- NULL
#names(top20_cond) <- c("%IncMSE", "names")

# optional: save top-20 variables with conditional importance as .csv
#write.csv(top20_cond,"results/top20_cond.rf_covid.count_full.csv",
#          row.names = FALSE)


# minimal model

# subset train and test data to top 20 identified by full model
df.train <- df.train[,names(df.train) %in% top20.vec]
df.test <- df.test[,names(df.test) %in% top20.vec]

# add protest to training and test data
df.train$protest <- y_train
df.test$protest <- y_test

# run grid search
modellist <- list()
for (ntree in ntrees) {
  fit = train(
    protest ~ .,
    tuneLength = 5,
    importance = TRUE,
    data = df.train,
    method = "rf",
    trControl = control,
    tuneGrid = tunegrid,
    ntree = ntree)
  key <- toString(ntree)
  modellist[[key]] <- fit
}
# optional: save scan configuration
#save(modellist,file="rf_optimization/covid.count_min_modellist.Rda")

# compare results
results <- resamples(modellist)
summary(results)
pdf(file = "figures/rf_covid.count_min.pdf", width = 6, height = 3.5)
```

```r
dotplot(results)
dev.off()

# select best model based on smallest RMSE/MAE and greatest Rsquared
ntree_opt <- "1000"
model_rf <- modellist[[ntree_opt]]

# save best model as .Rda for cross-prediction in section 5
model_rf_covid.count_min <- modellist[[ntree_opt]]
save(model_rf_covid.count_min, file="results/model_rf_covid.count_min.Rda")

# in-sample prediction: model fit
rf.fitted <- predict(model_rf)

# out-of-sample prediction: model performance
rf.predicted <- predict(model_rf, df.test)

# calculate performance statistics ("in" and "out" of sample)
performance <- list(
  RMSE.in = rmse(df.train$protest, rf.fitted),
  cor.in = cor(df.train$protest, rf.fitted),
  R2.in = cor(df.train$protest, rf.fitted)^2,
  MAE.in = mae(df.train$protest, rf.fitted),
  RMSE.out = rmse(df.test$protest, rf.predicted),
  cor.out = cor(df.test$protest, rf.predicted),
  R2.out = cor(df.test$protest, rf.predicted)^2,
  MAE.out = mae(df.test$protest, rf.predicted))

# save performance statistics
performance_rf_covid.count_min <- performance
save(performance_rf_covid.count_min,
     file = "results/performance_rf_covid.count_min.Rda")
```

4.1.2 Non-Covid protest with spatial split

```r
# subset to non-Covid protest
df <- df_save
df <- df[, c(-1:-4)]

# rename protest variable
colnames(df)[1] = "protest"

# one-hot encoding of categorical variables
oneHot.model <- dummyVars(" ~ .", data=df)
df <- predict(oneHot.model, newdata = df)
df <- data.frame(df)

# locality-based split (80/20)
# splits the data into training and testing sets, ensuring that all
# observations from a local government are grouped into the same subset
train.rows <- CreateLocalityPartition(df[, 1], locality = locality, p=0.8)
df.train <- df[train.rows,]
df.test <- df[-train.rows,]
```

```r
# save df.test as .csv for cross-prediction in section 5
write.csv(df.test,"results/df.test.cross_rf_non_covid.csv",
          row.names = FALSE)

# pull out protest from training and test data
y_train <- df.train[, 1]
y_test <- df.test[, 1]

# define number of folds on protest
folds.10 <- createFolds(y_train, k = 10)

# specify 10 folds and repeat 5 times
control <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 5,
  verboseIter = TRUE,
  savePredictions = "final",
  search="grid",
  index = folds.10)

# create tune grid with values from 3:12 for mtry
tunegrid <- expand.grid(.mtry=c(3:12))

# define ntrees, i.e. the number of trees to grow
ntrees <- c(50,80,100,200,500,1000,1500,2000)

# run grid search
modellist <- list()
for (ntree in ntrees) {
  fit = train(
    protest ~ .,
    tuneLength = 5,
    importance = TRUE,
    data = df.train,
    method = "rf",
    trControl = control,
    tuneGrid = tunegrid,
    ntree = ntree)
  key <- toString(ntree)
  modellist[[key]] <- fit
}
# optional: save scan configuration
#save(modellist,file="rf_optimization/non_covid.count_full_modellist.Rda")

# compare results
results <- resamples(modellist)
summary(results)
pdf(file = "figures/rf_non_covid.count_full.pdf", width = 6, height = 3.5)
dotplot(results)
dev.off()

# select best model based on smallest RMSE/MAE and greatest Rsquared
```

```r
ntree_opt <- "500"
model_rf <- modellist[[ntree_opt]]
mtry_opt <- as.numeric(unlist(model_rf$bestTune))

# optional: save best model as .Rda
#model_rf_non_covid.count_full <- modellist[[ntree_opt]]
#save(model_rf_non_covid.count_full,
#     file="results/model_rf_non_covid.count_full.Rda")

# in-sample prediction: model fit
rf.fitted <- predict(model_rf)

# out-of-sample prediction: model performance
rf.predicted <- predict(model_rf, df.test)

# calculate performance statistics ("in" and "out" of sample)
performance <- list(
  RMSE.in = rmse(df.train$protest, rf.fitted),
  cor.in = cor(df.train$protest, rf.fitted),
  R2.in = cor(df.train$protest, rf.fitted)^2,
  MAE.in = mae(df.train$protest, rf.fitted),
  RMSE.out = rmse(df.test$protest, rf.predicted),
  cor.out = cor(df.test$protest, rf.predicted),
  R2.out = cor(df.test$protest, rf.predicted)^2,
  MAE.out = mae(df.test$protest, rf.predicted))

# save performance statistics
performance_rf_non_covid.count_full <- performance
save(performance_rf_non_covid.count_full,
     file = "results/performance_rf_non_covid.count_full.Rda")


# variable permutation importance

# rerun classification using cforest from party pkg
rf.corr <- party::cforest(
  protest ~ .,
  data = df.train,
  control = cforest_unbiased(mtry = mtry_opt, ntree = as.numeric(ntree_opt))
)

# calculate unconditional variable importance
# (using faster implementation of permimp pkg)
varImp <- permimp::permimp(rf.corr, conditional = FALSE, asParty = TRUE)
varImp <- sort(varImp, decreasing = TRUE)

# get top-20 variables
top20 <- data.frame(varImp[c(1:20)])
top20$names <- row.names(top20)
row.names(top20) <- NULL
names(top20) <- c("%IncMSE", "names")
top20.vec <- top20$names
```

```r
# save top-20 variables as .csv (used in section 4.3)
write.csv(top20,"results/top20.rf_non_covid.count_full.csv",
          row.names = FALSE)

# optional:
# calculate conditional variable importance
# (using faster implementation of permimp pkg)
#.          [Note: Very long compute times!]
#varImp_cond <- permimp::permimp(rf.corr, conditional = TRUE)
#varImp_cond <- sort(varImp_cond$values, decreasing = TRUE)
#save(varImp_cond, file="results/varImp_cond.rf_non_covid.count_full.Rda")

# optional: get top-20 variables
#top20_cond <- data.frame(varImp_cond[c(1:20)])
#top20_cond$names <- row.names(top20_cond)
#row.names(top20_cond) <- NULL
#names(top20_cond) <- c("%IncMSE", "names")

# optional: save top-20 variables with conditional importance as .csv
#write.csv(top20_cond,"results/top20_cond.rf_non_covid.count_full.csv",
#          row.names = FALSE)


# minimal model

# subset train and test data to top 20 identified by full model
df.train <- df.train[,names(df.train) %in% top20.vec]
df.test <- df.test[,names(df.test) %in% top20.vec]

# add protest to training and test data
df.train$protest <- y_train
df.test$protest <- y_test

# run grid search
modellist <- list()
for (ntree in ntrees) {
  fit = train(
    protest ~ .,
    tuneLength = 5,
    importance = TRUE,
    data = df.train,
    method = "rf",
    trControl = control,
    tuneGrid = tunegrid,
    ntree = ntree)
  key <- toString(ntree)
  modellist[[key]] <- fit
}
# optional: save scan configuration
#save(modellist,file="rf_optimization/non_covid.count_min_modellist.Rda")

# compare results
results <- resamples(modellist)
```

```
summary(results)
pdf(file = "figures/rf_non_covid.count_min.pdf", width=6, height=3.5)
dotplot(results)
dev.off()

# select best model based on smallest RMSE/MAE and greatest Rsquared
ntree_opt <- "200"
model_rf <- modellist[[ntree_opt]]

# save best model as .Rda for cross-prediction in section 5
model_rf_non_covid.count_min <- modellist[[ntree_opt]]
save(model_rf_non_covid.count_min,
     file="results/model_rf_non_covid.count_min.Rda")

# in-sample prediction: model fit
rf.fitted <- predict(model_rf)

# out-of-sample prediction: model performance
rf.predicted <- predict(model_rf, df.test)

# calculate performance statistics ("in" and "out" of sample)
performance <- list(
  RMSE.in = rmse(df.train$protest, rf.fitted),
  cor.in = cor(df.train$protest, rf.fitted),
  R2.in = cor(df.train$protest, rf.fitted)^2,
  MAE.in = mae(df.train$protest, rf.fitted),
  RMSE.out = rmse(df.test$protest, rf.predicted),
  cor.out = cor(df.test$protest, rf.predicted),
  R2.out = cor(df.test$protest, rf.predicted)^2,
  MAE.out = mae(df.test$protest, rf.predicted))

# save performance statistics
performance_rf_non_covid.count_min <- performance
save(performance_rf_non_covid.count_min,
     file = "results/performance_rf_non_covid.count_min.Rda")
```

4.1.3 Covid protest with temporal split

```
# subset to Covid protest
df <- df_save
df <- df[, c(-1:-3, -5)]

# rename protest variable
colnames(df)[1] = "protest"

# one-hot encoding of categorical variables
oneHot.model <- dummyVars(" ~ .", data=df)
df <- predict(oneHot.model, newdata = df)
df <- data.frame(df)

# temporal split sample
# splits the data after 95 weeks into training and test set
df.train <- df[week <= 95,]
```

```
df.test <- df[week > 95,]

# pull out protest from training and test data
y_train <- df.train[, 1]
y_test <- df.test[, 1]

# define number of folds on protest
folds.10 <- createFolds(y_train, k = 10)

# specify 10 folds and repeat 5 times
control <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 5,
  verboseIter = TRUE,
  savePredictions = "final",
  search="grid",
  index = folds.10)

# create tune grid with values from 3:12 for mtry
tunegrid <- expand.grid(.mtry=c(3:12))

# define ntrees, i.e. the number of trees to grow
ntrees <- c(50,80,100,200,500,1000,1500,2000)

# run grid search
modellist <- list()
for (ntree in ntrees) {
  fit = train(
    protest ~ .,
    tuneLength = 5,
    importance = TRUE,
    data = df.train,
    method = "rf",
    trControl = control,
    tuneGrid = tunegrid,
    ntree = ntree)
  key <- toString(ntree)
  modellist[[key]] <- fit
}
# optional: save scan configuration
#save(modellist,file="rf_optimization/covid.count.temp_full_modellist.Rda")

# compare results
results <- resamples(modellist)
summary(results)
pdf(file = "figures/rf_covid.count.temp_full.pdf", width=6, height=3.5)
dotplot(results)
dev.off()

# select best model based on smallest RMSE/MAE and greatest Rsquared
ntree_opt <- "500"
model_rf <- modellist[[ntree_opt]]
```

```r
mtry_opt <- as.numeric(unlist(model_rf$bestTune))

# optional: save best model as .Rda
#model_rf_covid.count.temp_full <- modellist[[ntree_opt]]
#save(model_rf_covid.count.temp_full,
#     file="results/model_rf_covid.count.temp_full.Rda")

# in-sample prediction: model fit
rf.fitted <- predict(model_rf)

# out-of-sample prediction: model performance
rf.predicted <- predict(model_rf, df.test)

# calculate performance statistics ("in" and "out" of sample)
performance <- list(
  RMSE.in = rmse(df.train$protest, rf.fitted),
  cor.in = cor(df.train$protest, rf.fitted),
  R2.in = cor(df.train$protest, rf.fitted)^2,
  MAE.in = mae(df.train$protest, rf.fitted),
  RMSE.out = rmse(df.test$protest, rf.predicted),
  cor.out = cor(df.test$protest, rf.predicted),
  R2.out = cor(df.test$protest, rf.predicted)^2,
  MAE.out = mae(df.test$protest, rf.predicted))

# save performance statistics
performance_rf_covid.count.temp_full <- performance
save(performance_rf_covid.count.temp_full,
     file = "results/performance_rf_covid.count.temp_full.Rda")


# variable permutation importance

# rerun classification using cforest from party pkg
rf.corr <- party::cforest(
  protest ~ .,
  data = df.train,
  control = cforest_unbiased(mtry = mtry_opt, ntree = as.numeric(ntree_opt))
)

# calculate unconditional variable importance
# (using faster implementation of permimp pkg)
varImp <- permimp::permimp(rf.corr, conditional = FALSE, asParty = TRUE)
varImp <- sort(varImp$values, decreasing = TRUE)

# get top-20 variables
top20 <- data.frame(varImp[c(1:20)])
top20$names <- row.names(top20)
row.names(top20) <- NULL
names(top20) <- c("%IncMSE", "names")
top20.vec <- top20$names

# save top-20 variables as .csv (used for appendix figure)
write.csv(top20,"results/top20.rf_covid.count.temp_full.csv",
```

```
          row.names = FALSE)


# minimal model

# subset train and test data to top 20 identified by full model
df.train <- df.train[,names(df.train) %in% top20.vec]
df.test <- df.test[,names(df.test) %in% top20.vec]

# add protest to training and test data
df.train$protest <- y_train
df.test$protest <- y_test

# run grid search
modellist <- list()
for (ntree in ntrees) {
  fit = train(
    protest ~ .,
    tuneLength = 5,
    importance = TRUE,
    data = df.train,
    method = "rf",
    trControl = control,
    tuneGrid = tunegrid,
    ntree = ntree)
  key <- toString(ntree)
  modellist[[key]] <- fit
}
# optional: save scan configuration
#save(modellist,file="rf_optimization/covid.count.temp_min_modellist.Rda")

# compare results
results <- resamples(modellist)
summary(results)
pdf(file = "figures/rf_covid.count.temp_min.pdf", width=6, height=3.5)
dotplot(results)
dev.off()

# select best model based on smallest RMSE/MAE and greatest Rsquared
ntree_opt <- "100"
model_rf <- modellist[[ntree_opt]]

# optional: save best model as .Rda
#model_rf_non_covid.count_min <- modellist[[ntree_opt]]
#save(model_rf_non_covid.count_min,
     #file="results/model_rf_covid.count.temp_min.Rda")

# in-sample prediction: model fit
rf.fitted <- predict(model_rf)

# out-of-sample prediction: model performance
rf.predicted <- predict(model_rf, df.test)
```

```r
# calculate performance statistics ("in" and "out" of sample)
performance <- list(
  RMSE.in = rmse(df.train$protest, rf.fitted),
  cor.in = cor(df.train$protest, rf.fitted),
  R2.in = cor(df.train$protest, rf.fitted)^2,
  MAE.in = mae(df.train$protest, rf.fitted),
  RMSE.out = rmse(df.test$protest, rf.predicted),
  cor.out = cor(df.test$protest, rf.predicted),
  R2.out = cor(df.test$protest, rf.predicted)^2,
  MAE.out = mae(df.test$protest, rf.predicted))

# save performance statistics
performance_rf_covid.count.temp_min <- performance
save(performance_rf_covid.count.temp_min,
     file = "results/performance_rf_covid.count.temp_min.Rda")
```

4.1.4 Non-Covid protest with temporal split

```r
# subset to non-Covid protest
df <- df_save
df <- df[, c(-1:-4)]

# rename protest variable
colnames(df)[1] = "protest"

# one-hot encoding of categorical variables
oneHot.model <- dummyVars(" ~ .", data=df)
df <- predict(oneHot.model, newdata = df)
df <- data.frame(df)

# temporal split sample
# splits the data after 95 weeks into training and test set
df.train <- df[week <= 95,]
df.test <- df[week > 95,]

# pull out protest from training and test data
y_train <- df.train[, 1]
y_test <- df.test[, 1]

# define number of folds on protest
folds.10 <- createFolds(y_train, k = 10)

# specify 10 folds and repeat 5 times
control <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 5,
  verboseIter = TRUE,
  savePredictions = "final",
  search="grid",
  index = folds.10)

# create tune grid with values from 3:12 for mtry
```

```r
tunegrid <- expand.grid(.mtry=c(3:12))

# define ntrees, i.e. the number of trees to grow
ntrees <- c(50,80,100,200,500,1000,1500,2000)

# run grid search
modellist <- list()
for (ntree in ntrees) {
  fit = train(
    protest ~ .,
    tuneLength = 5,
    importance = TRUE,
    data = df.train,
    method = "rf",
    trControl = control,
    tuneGrid = tunegrid,
    ntree = ntree)
  key <- toString(ntree)
  modellist[[key]] <- fit
}
# optional: save scan configuration
#save(modellist,file="rf_optimization/non_covid.count.temp_full.Rda")

# compare results
results <- resamples(modellist)
summary(results)
pdf(file = "figures/rf_non_covid.count.temp_full.pdf", width=6, height=3.5)
dotplot(results)
dev.off()

# select best model based on smallest RMSE/MAE and greatest Rsquared
ntree_opt <- "200"
model_rf <- modellist[[ntree_opt]]
mtry_opt <- as.numeric(unlist(model_rf$bestTune))

# optional: save best model as .Rda
#model_rf_non_covid.count.temp_full <- modellist[[ntree_opt]]
#save(model_rf_non_covid.count.temp_full,
#     file="results/model_rf_non_covid.count.temp_full.Rda")

# in-sample prediction: model fit
rf.fitted <- predict(model_rf)

# out-of-sample prediction: model performance
rf.predicted <- predict(model_rf, df.test)

# calculate performance statistics ("in" and "out" of sample)
performance <- list(
  RMSE.in = rmse(df.train$protest, rf.fitted),
  cor.in = cor(df.train$protest, rf.fitted),
  R2.in = cor(df.train$protest, rf.fitted)^2,
  MAE.in = mae(df.train$protest, rf.fitted),
  RMSE.out = rmse(df.test$protest, rf.predicted),
```

```
  cor.out = cor(df.test$protest, rf.predicted),
  R2.out = cor(df.test$protest, rf.predicted)^2,
  MAE.out = mae(df.test$protest, rf.predicted))

# save performance statistics
performance_rf_non_covid.count.temp_full <- performance
save(performance_rf_non_covid.count.temp_full,
     file = "results/performance_rf_non_covid.count.temp_full.Rda")


# variable permutation importance

# rerun classification using cforest from party pkg
rf.corr <- party::cforest(
  protest ~ .,
  data = df.train,
  control = cforest_unbiased(mtry = mtry_opt, ntree = as.numeric(ntree_opt))
)

# calculate unconditional variable importance
# (using faster implementation of permimp pkg)
varImp <- permimp::permimp(rf.corr, conditional = FALSE, asParty = TRUE)
varImp <- sort(varImp$values, decreasing = TRUE)

# get top-20 variables
top20 <- data.frame(varImp[c(1:20)])
top20$names <- row.names(top20)
row.names(top20) <- NULL
names(top20) <- c("%IncMSE", "names")
top20.vec <- top20$names

# save top-20 variables as .csv (used for appendix figure)
write.csv(top20,"results/top20.rf_non_covid.count.temp_full.csv",
          row.names = FALSE)


# minimal model

# subset train and test data to top 20 identified by full model
df.train <- df.train[,names(df.train) %in% top20.vec]
df.test <- df.test[,names(df.test) %in% top20.vec]

# add protest to training and test data
df.train$protest <- y_train
df.test$protest <- y_test

# run grid search
modellist <- list()
for (ntree in ntrees) {
  fit = train(
    protest ~ .,
    tuneLength = 5,
    importance = TRUE,
```

```
    data = df.train,
    method = "rf",
    trControl = control,
    tuneGrid = tunegrid,
    ntree = ntree)
  key <- toString(ntree)
  modellist[[key]] <- fit
}
# optional: save scan configuration
#save(modellist,file="rf_optimization/non_covid.count.temp_min_modellist.Rda")

# compare results
results <- resamples(modellist)
summary(results)
pdf(file = "figures/rf_non_covid.count.temp_min.pdf", width=6, height=3.5)
dotplot(results)
dev.off()

# select best model based on smallest RMSE/MAE and greatest Rsquared
ntree_opt <- "200"
model_rf <- modellist[[ntree_opt]]

# optional: save best model as .Rda
#model_rf_non_covid.count_min <- modellist[[ntree_opt]]
#save(model_rf_non_covid.count_min,
#     file="results/model_rf_non_covid.count.temp_min.Rda")

# in-sample prediction: model fit
rf.fitted <- predict(model_rf)

# out-of-sample prediction: model performance
rf.predicted <- predict(model_rf, df.test)

# calculate performance statistics ("in" and "out" of sample)
performance <- list(
  RMSE.in = rmse(df.train$protest, rf.fitted),
  cor.in = cor(df.train$protest, rf.fitted),
  R2.in = cor(df.train$protest, rf.fitted)^2,
  MAE.in = mae(df.train$protest, rf.fitted),
  RMSE.out = rmse(df.test$protest, rf.predicted),
  cor.out = cor(df.test$protest, rf.predicted),
  R2.out = cor(df.test$protest, rf.predicted)^2,
  MAE.out = mae(df.test$protest, rf.predicted))

# save performance statistics
performance_rf_non_covid.count.temp_min <- performance
save(performance_rf_non_covid.count.temp_min,
     file = "results/performance_rf_non_covid.count.temp_min.Rda")
```

4.2 Random forest binary classification

Again, we use a spatial split to look at Covid protest (Section 4.2.1), then non-Covid protest outcomes (Section 4.2.2), this time using a random forest classification model with a binary protest outcome (yes/no).

Note that we also calculate a full model and minimal model (with only the top-20 predictors) for each case, using cross-valiation.

4.2.1 Covid protest with spatial split

```
# subset to Covid protest
df <- df_save
df <- df[, c(-1:-3, -5)]

# rename protest variable
colnames(df)[1] = "protest"

# code binary outcome variable
df$protest <- as.factor(ifelse(df$protest  >= 1, "1", "0"))

# locality-based split (80/20)
# splits the data into training and testing sets, ensuring that all
# observations from a local government are grouped into the same subset
train.rows <- CreateLocalityPartition(df[, 1], locality = locality, p=0.8)
df.train <- df[train.rows,]
df.test <- df[-train.rows,]

# pull out protest from training and test data
y_train <- df.train[, 1]
y_test <- df.test[, 1]

# define number of folds on protest
folds.10 <- createFolds(y_train, k = 10)

# specify 10 folds and repeat 5 times
control <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 5,
  verboseIter = TRUE,
  savePredictions = "final",
  search="grid",
  index = folds.10)

# create tune grid with values from 3:12 for mtry
tunegrid <- expand.grid(.mtry=c(3:12))

# define ntrees, i.e. the number of trees to grow
ntrees <- c(50,80,100,200,500,1000,1500,2000)

# run grid search
modellist <- list()
for (ntree in ntrees) {
  fit = train(
    protest ~ .,
    tuneLength = 5,
    importance = TRUE,
    data = df.train,
    method = "rf",
```

```
    trControl = control,
    tuneGrid = tunegrid,
    ntree = ntree)
  key <- toString(ntree)
  modellist[[key]] <- fit
}
# optional: save scan configuration
#save(modellist,file="rf_optimization/covid.bin_full_modellist.Rda")

# compare results
results <- resamples(modellist)
summary(results)
pdf(file = "figures/rf_covid.bin_full.pdf", width = 6, height = 3.5)
dotplot(results)
dev.off()

# select best model based on greatest accuracy and lowest kappa
ntree_opt <- "2000"
model_rf <- modellist[[ntree_opt]]
mtry_opt <- as.numeric(unlist(model_rf$bestTune))

# optional: save best model as .Rda
#model_rf_covid.bin_full <- modellist[[ntree_opt]]
#save(model_rf_covid.bin_full, file="results/model_rf_covid.bin_full.Rda")

# in-sample prediction: model fit
rf.fitted <- predict(model_rf)

# out-of-sample prediction: model performance
rf.predicted <- predict(model_rf, df.test)

# confusion matrix: in-sample
cm.in <- confusionMatrix(reference = df.train$protest,
                         data = rf.fitted,
                         positive = "1",
                         mode = "prec_recall")

# confusion matrix: out-of-sample
cm.out <- confusionMatrix(reference = df.test$protest,
                         data = rf.predicted,
                         positive = "1",
                         mode = "prec_recall")

# save performance statistics ("in" and "out" of sample)
performance_rf_covid.bin_full <- list(in_sample = cm.in, out_sample = cm.out)
save(performance_rf_covid.bin_full,
     file = "results/performance_rf_covid.bin_full.Rda")


# variable permutation importance

# rerun classification using cforest from party pkg
rf.corr <- party::cforest(
```

```
  protest ~ .,
  data = df.train,
  control = cforest_unbiased(mtry = mtry_opt, ntree = as.numeric(ntree_opt))
)

# calculate unconditional variable importance
# (using faster implementation of permimp pkg)
varImp <- permimp::permimp(rf.corr, conditional = FALSE, asParty = TRUE)
varImp <- sort(varImp$values, decreasing = TRUE)

# get top-20 variables
top20 <- data.frame(varImp[c(1:20)])
top20$names <- row.names(top20)
row.names(top20) <- NULL
names(top20) <- c("%IncMSE", "names")
top20.vec <- top20$names

# save top-20 variables as .csv (used for appendix figure)
write.csv(top20,"results/top20.rf_covid.bin_full.csv", row.names = FALSE)


# minimal model

# subset train and test data to top 20 identified by full model
df.train <- df.train[,names(df.train) %in% top20.vec]
df.test <- df.test[,names(df.test) %in% top20.vec]

# add protest to training and test data
df.train$protest <- y_train
df.test$protest <- y_test

# run grid search
modellist <- list()
for (ntree in ntrees) {
  fit = train(
    protest ~ .,
    tuneLength = 5,
    importance = TRUE,
    data = df.train,
    method = "rf",
    trControl = control,
    tuneGrid = tunegrid,
    ntree = ntree)
  key <- toString(ntree)
  modellist[[key]] <- fit
}
# optional: save scan configuration
#save(modellist,file="rf_optimization/covid.bin_min_modellist.Rda")

# compare results
results <- resamples(modellist)
summary(results)
pdf(file = "figures/rf_covid.bin_min.pdf", width=6, height=3.5)
```

```r
dotplot(results)
dev.off()

# select best model based on greatest accuracy and lowest kappa
ntree_opt <- "200"
model_rf <- modellist[[ntree_opt]]

# optional: save best model as .Rda
#model_rf_covid.bin_min <- modellist[[ntree_opt]]
#save(model_rf_covid.bin_min, file="results/model_rf_covid.bin_min.Rda")

# in-sample prediction: model fit
rf.fitted <- predict(model_rf)

# out-of-sample prediction: model performance
rf.predicted <- predict(model_rf, df.test)

# confusion matrix: in-sample
cm.in <- confusionMatrix(reference = df.train$protest,
                         data = rf.fitted,
                         positive = "1",
                         mode = "prec_recall")

# confusion matrix: out-of-sample
cm.out <- confusionMatrix(reference = df.test$protest,
                         data = rf.predicted,
                         positive = "1",
                         mode = "prec_recall")

# save performance statistics ("in" and "out" of sample)
performance_rf_covid.bin_min <- list(in_sample = cm.in, out_sample = cm.out)
save(performance_rf_covid.bin_min,
     file = "results/performance_rf_covid.bin_min.Rda")
```

4.2.2 Non-Covid protest with spatial split

```r
# subset to non-Covid protest
df <- df_save
df <- df[, c(-1:-4)]

# rename protest variable
colnames(df)[1] = "protest"

# code binary outcome variable
df$protest <- as.factor(ifelse(df$protest  >= 1, "1", "0"))

# locality-based split (80/20)
# splits the data into training and testing sets, ensuring that all
# observations from a local government are grouped into the same subset
train.rows <- CreateLocalityPartition(df[, 1], locality = locality, p=0.8)
df.train <- df[train.rows,]
df.test <- df[-train.rows,]
```

```r
# pull out protest from training and test data
y_train <- df.train[, 1]
y_test <- df.test[, 1]

# define number of folds on protest
folds.10 <- createFolds(y_train, k = 10)

# specify 10 folds and repeat 5 times
control <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 5,
  verboseIter = TRUE,
  savePredictions = "final",
  search="grid",
  index = folds.10)

# create tune grid with values from 3:12 for mtry
tunegrid <- expand.grid(.mtry=c(3:12))

# define ntrees, i.e. the number of trees to grow
ntrees <- c(50,80,100,200,500,1000,1500,2000)

# run grid search
modellist <- list()
for (ntree in ntrees) {
  fit = train(
    protest ~ .,
    tuneLength = 5,
    importance = TRUE,
    data = df.train,
    method = "rf",
    trControl = control,
    tuneGrid = tunegrid,
    ntree = ntree)
  key <- toString(ntree)
  modellist[[key]] <- fit
}
# optional: save scan configuration
#save(modellist,file="rf_optimization/non_covid.bin_full_modellist.Rda")

# compare results
results <- resamples(modellist)
summary(results)
pdf(file = "figures/rf_non_covid.bin_full.pdf", width = 6, height = 3.5)
dotplot(results)
dev.off()

# select best model based on greatest accuracy and lowest kappa
ntree_opt <- "1000"
model_rf <- modellist[[ntree_opt]]
mtry_opt <- as.numeric(unlist(model_rf$bestTune))
```

```r
# optional: save best model as .Rda
#model_rf_non_covid.bin_full <- modellist[[ntree_opt]]
#save(model_rf_non_covid.bin_full,
#     file="results/model_rf_non_covid.bin_full.Rda")

# in-sample prediction: model fit
rf.fitted <- predict(model_rf)

# out-of-sample prediction: model performance
rf.predicted <- predict(model_rf, df.test)

# confusion matrix: in-sample
cm.in <- confusionMatrix(reference = df.train$protest,
                         data = rf.fitted,
                         positive = "1",
                         mode = "prec_recall")

# confusion matrix: out-of-sample
cm.out <- confusionMatrix(reference = df.test$protest,
                         data = rf.predicted,
                         positive = "1",
                         mode = "prec_recall")

# save performance statistics ("in" and "out" of sample)
performance_rf_non_covid.bin_full <- list(in_sample = cm.in,
                                          out_sample = cm.out)
save(performance_rf_non_covid.bin_full,
     file = "results/performance_rf_non_covid.bin_full.Rda")


# variable permutation importance

# rerun classification using cforest from party pkg
rf.corr <- party::cforest(
  protest ~ .,
  data = df.train,
  control = cforest_unbiased(mtry = mtry_opt, ntree = as.numeric(ntree_opt))
)

# calculate unconditional variable importance
# (using faster implementation of permimp pkg)
varImp <- permimp::permimp(rf.corr, conditional = FALSE, asParty = TRUE)
varImp <- sort(varImp$values, decreasing = TRUE)

# get top-20 variables
top20 <- data.frame(varImp[c(1:20)])
top20$names <- row.names(top20)
row.names(top20) <- NULL
names(top20) <- c("%IncMSE", "names")
top20.vec <- top20$names

# save top-20 variables as .csv (used for appendix figure)
write.csv(top20,"results/top20.rf_non_covid.bin_full.csv",
```

```
          row.names = FALSE)


# minimal model

# subset train and test data to top 20 identified by full model
df.train <- df.train[,names(df.train) %in% top20.vec]
df.test <- df.test[,names(df.test) %in% top20.vec]

# add protest to training and test data
df.train$protest <- y_train
df.test$protest <- y_test

# run grid search
modellist <- list()
for (ntree in ntrees) {
  fit = train(
    protest ~ .,
    tuneLength = 5,
    importance = TRUE,
    data = df.train,
    method = "rf",
    trControl = control,
    tuneGrid = tunegrid,
    ntree = ntree)
  key <- toString(ntree)
  modellist[[key]] <- fit
}
# optional: save scan configuration
#save(modellist,file="rf_optimization/non_covid.bin_min_modellist.Rda")

# compare results
results <- resamples(modellist)
summary(results)
pdf(file = "figures/rf_non_covid.bin_min.pdf", width=6, height=3.5)
dotplot(results)
dev.off()

# select best model based on greatest accuracy and lowest kappa
ntree_opt <- "500"
model_rf <- modellist[[ntree_opt]]

# optional: save best model as .Rda
#model_rf_non_covid.bin_min <- modellist[[ntree_opt]]
#save(model_rf_non_covid.bin_min,
#     file="results/model_rf_non_covid.bin_min.Rda")

# in-sample prediction: model fit
rf.fitted <- predict(model_rf)

# out-of-sample prediction: model performance
rf.predicted <- predict(model_rf, df.test)
```

```r
# confusion matrix: in-sample
cm.in <- confusionMatrix(reference = df.train$protest,
                         data = rf.fitted,
                         positive = "1",
                         mode = "prec_recall")

# confusion matrix: out-of-sample
cm.out <- confusionMatrix(reference = df.test$protest,
                         data = rf.predicted,
                         positive = "1",
                         mode = "prec_recall")

# save performance statistics ("in" and "out" of sample)
performance_rf_non_covid.bin_min <- list(in_sample = cm.in,
                                         out_sample = cm.out)
save(performance_rf_non_covid.bin_min,
     file = "results/performance_rf_non_covid.bin_min.Rda")
```

4.3 Comparison of top-20 random forest drivers for Covid and Non-Covid protests

This code generates Figure 3 from the manuscript. The figure illustrates the top-20 important drivers of protest using the spatial split sample, per type and based on the random forest regression model.

Since the random forest is not executed by default in this markdown file (due to the long run times), the inputs are imported from the saved .csv files provided in the "results" folder. The same is true for all subsequent (appendix) figures generated in this markdown.

```r
# import (sorted) top-20 Covid protest drivers
drivers.cov <- read.csv("results/top20.rf_covid.count_full.csv")
colnames(drivers.cov) <- c("value", "names")

# import (sorted) top-20 non-Covid protest drivers
drivers.noncov <- read.csv("results/top20.rf_non_covid.count_full.csv")
colnames(drivers.noncov) <- c("value", "names")

# convert to relative importance values (in %)
drivers.cov.total <- sum(drivers.cov$value)
drivers.cov$value <- -100*drivers.cov$value/drivers.cov.total
drivers.noncov.total <- sum(drivers.noncov$value)
drivers.noncov$value <- 100*drivers.noncov$value/drivers.noncov.total

# add missing rows from one df in the other
drivers <- merge(drivers.cov, drivers.noncov, by= c("names"),
                 all.x = TRUE, all.y = TRUE)

drivers.covid <- drivers[, c(1,2)]
colnames(drivers.covid)[2] ="value"
drivers.covid

drivers.noncov <- drivers[, c(1,3)]
colnames(drivers.noncov)[2] ="value"
drivers.noncov
```

```r
# add column with protest type
drivers.covid$type <- "Covid protests"
drivers.noncov$type <- "Non-Covid protests"

# replace NA with 0
drivers.covid[is.na(drivers.covid$value),]$value <- 0
drivers.noncov[is.na(drivers.noncov$value),]$value <- 0

# rbind
df.plot <- rbind(drivers.covid, drivers.noncov)
df.plot <- df.plot[!is.na(df.plot$names),]

# import list with variables and mechanisms
df.mechanism <- read.csv("data/raw_mechanism.csv")

# get mechanisms and variable names
df.plot <- merge(df.plot, df.mechanism, by="names", all.x=TRUE)
df.plot$type <- as.factor(df.plot$type)

# indicator levels
df.plot <- df.plot[order(df.plot$type, df.plot$value),]
lvls <- df.plot$indicator
lvls <- unique(lvls)
df.plot$indicator <- factor(df.plot$indicator, levels=c(lvls))

# mechanism levels
lvls.m <- c("Opportunity", "Coercion", "Structure", "Grievance")
df.plot$mechanism <- factor(df.plot$mechanism, levels=c(lvls.m))
df.plot <- df.plot[!is.na(df.plot$indicator),]

##### plot

fig3 <- ggplot2::ggplot() +
  geom_bar(data = df.plot,
           aes(x=value, y=fct_rev(indicator), fill= mechanism),
           stat = "identity", colour="black", linewidth= .1,
           width = .75, alpha = 0.9, na.rm = TRUE) +
  scale_fill_manual(name = "",
                    values = c("#de2414","#ff6161",
                               "#ff6699","#ffb3de"),
                    labels = c("Opportunity", "Coercion",
                               "Structure","Grievance"),
                    na.translate = F) +
  annotate("text", x = c(-7, 7), y = c(26.5,26.5),
           label = c("Covid protest", "Non-Covid protest"),
           size = 4.5, fontface = "bold") +
  ggplot2::labs(
    y = " ",
    x = "Variable importance (% relative decrease in model accuracy)") +
  scale_y_discrete(expand=c(0.1,0)) +
  scale_x_continuous(limits = c(-14.6, 14.6),
                     breaks=c(-14, -12, -10, -8, -6, -4, -2, 0,
                              2, 4, 6, 8, 10, 12, 14),
```

```
                    labels=c("14","12","10","8","6","4","2","0",
                             "2","4","6","8","10","12","14")) +
  theme_minimal() +
  theme(panel.grid.major.x = element_blank(),
        panel.grid.minor.x = element_blank(),
        panel.grid.minor.y = element_blank(),
        panel.grid.major.y = element_blank(),
        axis.text.x = element_text(size =10),
        axis.title.x  = element_text(size =12, face = "bold"),
        axis.text.y =  element_text(size =10),
        axis.title.y = element_text(size=12),
        legend.position= c(.5, 1.04),
        legend.direction = "horizontal",
        legend.text = element_text(size=12),
        plot.title = element_blank(),
        plot.subtitle = element_blank(),
        plot.background = element_blank(),
        plot.margin=unit(c(4, 0.5, 2, 0.5), units="line")) #t, r, b, l)

fig3
```

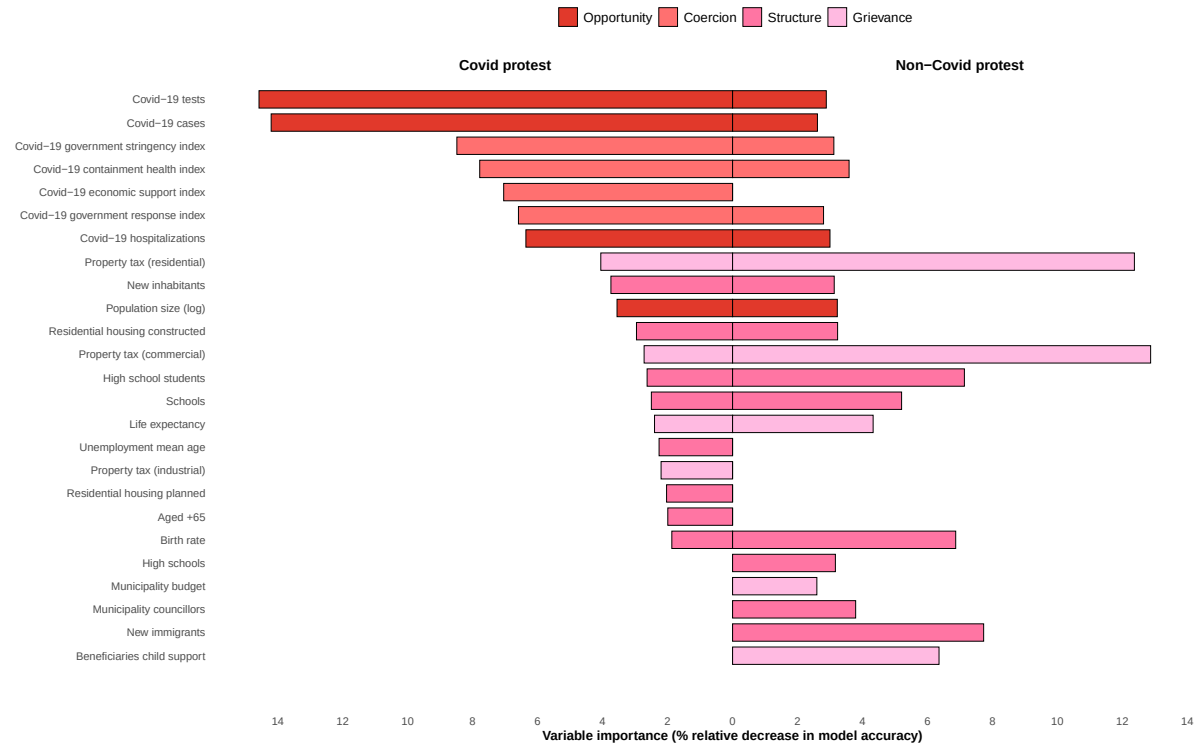

Figure 5: Top-20 important drivers of protest, per type based on the random forest classifier (Figure 3).

```
# optional: save Figure 3 as .svg or .pdf
#ggsave("figures/fig_3.svg", fig3,
#       width=16.79, height=10.65, units="in", dpi=300)
#ggsave("figures/fig_3.pdf", fig3, width=16.79,
#       height=10.65, units="in", dpi=300)
```

5. Cross-prediction

Cross-prediction is an additional technique to test our models' out-of-sample predictive power and generalization to independent data that was not used in estimating the models. In particular, we train Poisson regression and random forest regression models on Covid protest counts and then predict non-Covid protest, and vice versa.

```
# GLM

# import test data
df.test_covid <- read.csv("results/df.test.cross_glm_covid.csv")
df.test_non_covid <- read.csv("results/df.test.cross_glm_non_covid.csv")

# import models
load("results/model_glm_covid.count_min.Rda")
model_covid <- model_glm_covid.count_min$finalModel
load("results/model_glm_non_covid.count_min.Rda")
model_non_covid <- model_glm_non_covid.count_min$finalModel

# use trained Covid model to predict non-Covid protest

# out of sample prediction
glm.predicted <- predict(model_covid, df.test_non_covid)

# calculate performance statistics
performance <- list(
  RMSE.out = rmse(df.test_non_covid$protest, glm.predicted),
  cor.out = cor(df.test_non_covid$protest, glm.predicted),
  R2.out = cor(df.test_non_covid$protest, glm.predicted)^2,
  MAE.out = mae(df.test_non_covid$protest, glm.predicted))

# save performance statistics
performance_cross_glm_covid.count_full <- performance
save(performance_cross_glm_covid.count_full,
     file = "results/performance_cross_glm_covid.count_full.Rda")

# use trained non-Covid model to predict Covid protest

# out of sample prediction
glm.predicted <- predict(model_non_covid, df.test_covid)

# calculate performance statistics
performance <- list(
  RMSE.out = rmse(df.test_covid$protest, glm.predicted),
  cor.out = cor(df.test_covid$protest, glm.predicted),
  R2.out = cor(df.test_covid$protest, glm.predicted)^2,
  MAE.out = mae(df.test_covid$protest, glm.predicted))

# save performance statistics
performance_cross_glm_non_covid.count_full <- performance
save(performance_cross_glm_non_covid.count_full,
     file = "results/performance_cross_glm_non_covid.count_full.Rda")
```

```r
# Random forest

# import test data
df.test_covid <- read.csv("results/df.test.cross_rf_covid.csv")
df.test_non_covid <- read.csv("results/df.test.cross_rf_non_covid.csv")

# import models
load("results/model_rf_covid.count_min.Rda")
model_covid <- model_rf_covid.count_min$finalModel
load("results/model_rf_non_covid.count_min.Rda")
model_non_covid <- model_rf_covid.count_min$finalModel

# use trained Covid model to predict non-Covid protest

# out-of-sample prediction
rf.predicted <- predict(model_covid, df.test_non_covid)

# calculate performance statistics
performance <- list(
  RMSE.out = rmse(df.test_non_covid$protest, rf.predicted),
  cor.out = cor(df.test_non_covid$protest, rf.predicted),
  R2.out = cor(df.test_non_covid$protest, rf.predicted)^2,
  MAE.out = mae(df.test_non_covid$protest, rf.predicted))

# save performance statistics
performance_cross_rf_covid.count_full <- performance
save(performance_cross_rf_covid.count_full,
     file = "results/performance_cross_rf_covid.count_full.Rda")


# use trained non-Covid model to predict Covid protest

# out-of-sample prediction
rf.predicted <- predict(model_non_covid, df.test_covid)

# calculate performance statistics
performance <- list(
  RMSE.out = rmse(df.test_covid$protest, rf.predicted),
  cor.out = cor(df.test_covid$protest, rf.predicted),
  R2.out = cor(df.test_covid$protest, rf.predicted)^2,
  MAE.out = mae(df.test_covid$protest, rf.predicted))

# save performance statistics
performance_cross_rf_non_covid.count_full <- performance
save(performance_cross_rf_non_covid.count_full,
     file = "results/performance_cross_rf_non_covid.count_full.Rda")
```

Figure S9

Figure S9 illustrates the top-20 important drivers of protest using the temporal split sample, per type and based on the full random forest model.

```r
# import (sorted) top-20 Covid protest drivers
drivers.cov <- read.csv("results/top20.rf_covid.count.temp_full.csv")
```

```r
colnames(drivers.cov) <- c("value", "names")

# import (sorted) top-20 non-Covid protest drivers
drivers.noncov <- read.csv("results/top20.rf_non_covid.count.temp_full.csv")
colnames(drivers.noncov) <- c("value", "names")

# convert to relative importance values (in %)
drivers.cov.total <- sum(drivers.cov$value)
drivers.cov$value <- -100*drivers.cov$value/drivers.cov.total
drivers.noncov.total <- sum(drivers.noncov$value)
drivers.noncov$value <- 100*drivers.noncov$value/drivers.noncov.total

# add missing rows from one df in the other
drivers <- merge(drivers.cov, drivers.noncov, by= c("names"),
                 all.x = TRUE, all.y = TRUE)

drivers.covid <- drivers[, c(1,2)]
colnames(drivers.covid)[2] ="value"
drivers.covid

drivers.noncov <- drivers[, c(1,3)]
colnames(drivers.noncov)[2] ="value"
drivers.noncov

# add column with protest type
drivers.covid$type <- "Covid protests"
drivers.noncov$type <- "Non-Covid protests"

# replace NA with 0
drivers.covid[is.na(drivers.covid$value),]$value <- 0
drivers.noncov[is.na(drivers.noncov$value),]$value <- 0

# rbind
df.plot <- rbind(drivers.covid, drivers.noncov)
df.plot <- df.plot[!is.na(df.plot$names),]

# import list with variables and mechanisms
df.mechanism <- read.csv("data/raw_mechanism.csv")

# get mechanisms and variable names
df.plot <- merge(df.plot, df.mechanism, by="names", all.x=TRUE)
df.plot$type <- as.factor(df.plot$type)

# indicator levels
df.plot <- df.plot[order(df.plot$type, df.plot$value),]
lvls <- df.plot$indicator
lvls <- unique(lvls)
df.plot$indicator <- factor(df.plot$indicator, levels=c(lvls))

# mechanism levels
lvls.m <- c("Opportunity", "Coercion", "Structure", "Grievance")
df.plot$mechanism <- factor(df.plot$mechanism, levels=c(lvls.m))
df.plot <- df.plot[!is.na(df.plot$indicator),]
```

```
##### plot

fig_s09 <- ggplot2::ggplot() +
  geom_bar(data = df.plot,
           aes(x=value, y=fct_rev(indicator), fill= mechanism),
           stat = "identity", colour="black", linewidth= .1,
           width = .75, alpha = 0.9, na.rm = TRUE) +
  scale_fill_manual(name = "",
                    values = c("#de2414","#ff6161",
                               "#ff6699","#ffb3de"),
                    labels = c("Opportunity", "Coercion",
                               "Structure","Grievance"),
                    na.translate = F) +
  annotate("text", x = c(-6, 6), y = c(29.5,29.5),
           label = c("Covid protest", "Non-Covid protest"),
           size = 4.5, fontface = "bold") +
  ggplot2::labs(
    y = " ",
    x = "Variable importance (% relative decrease in model accuracy)") +
  scale_y_discrete(expand=c(0.1,0)) +
  scale_x_continuous(limits = c(-12.5, 12.4),
                     breaks=c(-12, -10, -8, -6, -4, -2, 0,
                              2, 4, 6, 8, 10, 12),
                     labels=c("12","10","8","6","4","2","0",
                              "2","4","6","8","10","12")) +
  theme_minimal() +
  theme(panel.grid.major.x = element_blank(),
        panel.grid.minor.x = element_blank(),
        panel.grid.minor.y = element_blank(),
        panel.grid.major.y = element_blank(),
        axis.text.x = element_text(size =10),
        axis.title.x  = element_text(size =12, face = "bold"),
        axis.text.y =  element_text(size =10),
        axis.title.y = element_text(size=12),
        legend.position= c(.5, 1.04),
        legend.direction = "horizontal",
        legend.text = element_text(size=12),
        plot.title = element_blank(),
        plot.subtitle = element_blank(),
        plot.background = element_blank(),
        plot.margin=unit(c(4, 0.5, 2, 0.5), units="line")) #t, r, b, l)

fig_s09
```

```
# optional: save Figure 3 as .svg or .pdf
#ggsave("figures/si_fig_s9.svg", fig_s09,
#       width=16.79, height=10.65, units="in", dpi=300)
#ggsave("figures/si_fig_s9.pdf", fig_s09, width=16.79,
#       height=10.65, units="in", dpi=300)
```

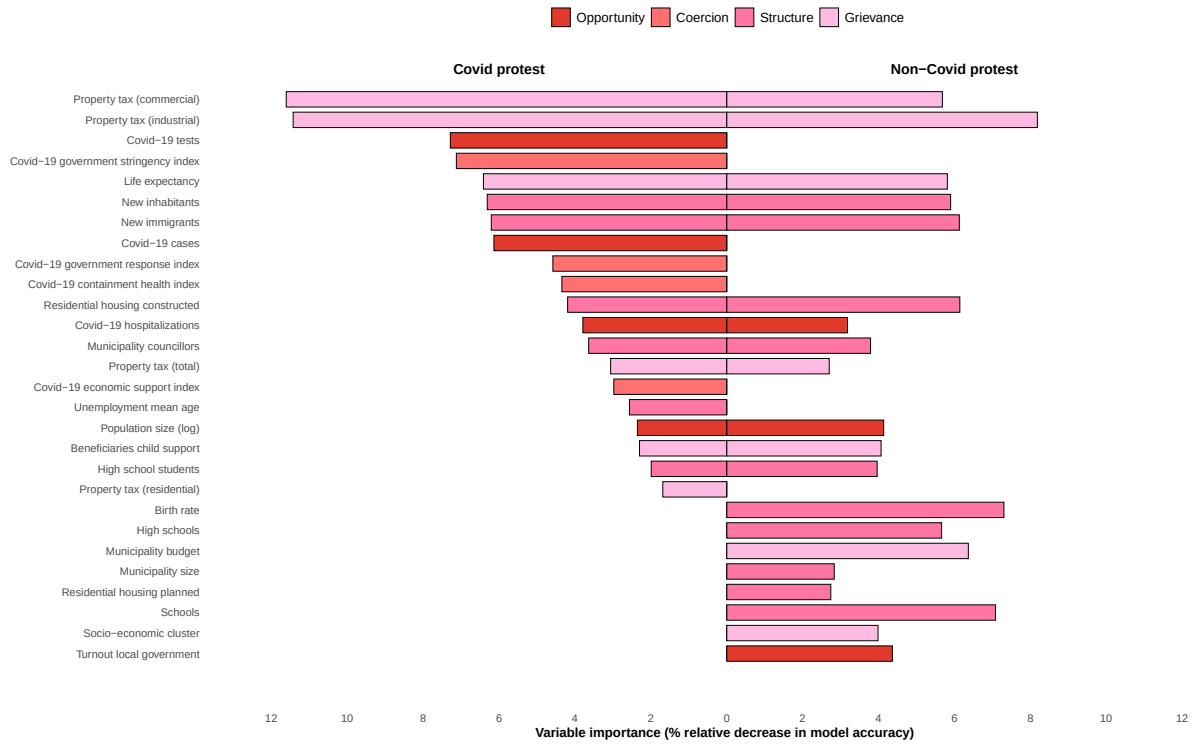

Figure 6: Temporal split: Top-20 important drivers of protest, per type and based on the full random forest model (Figure S9).

Figure S12

Figure S12 illustrates the top-20 important drivers of protest using the spatial split sample, per type and based on the binary random forest classification model.

6. Alternative sample

We run the main GLM and random forest analysis using the full sample of 255 Israeli local governments, i.e. municipalities including regional councils (`255_protest_locality-week.csv`), as an additional sensitivity check. The below analysis indicates robust performance, albeit slightly lower relative to the full model GLM and random forest models using 189 municipalities.

```
# import data
df_save <- read.csv("data/255_protest.locality-week.csv",
                    stringsAsFactors = TRUE)

# re-code locality_id to 1-255
df_save$locality_id <- as.character(df_save$locality_id)
df_save <- data.frame(df_save %>%
                        group_by(locality_id) %>%
                        dplyr::mutate(locality_id = cur_group_id()))

# log population size
df_save$x47 <- log(df_save$x47)

# change variable names
colnames(df_save)[6:64] <- c( "Jewish population",
```

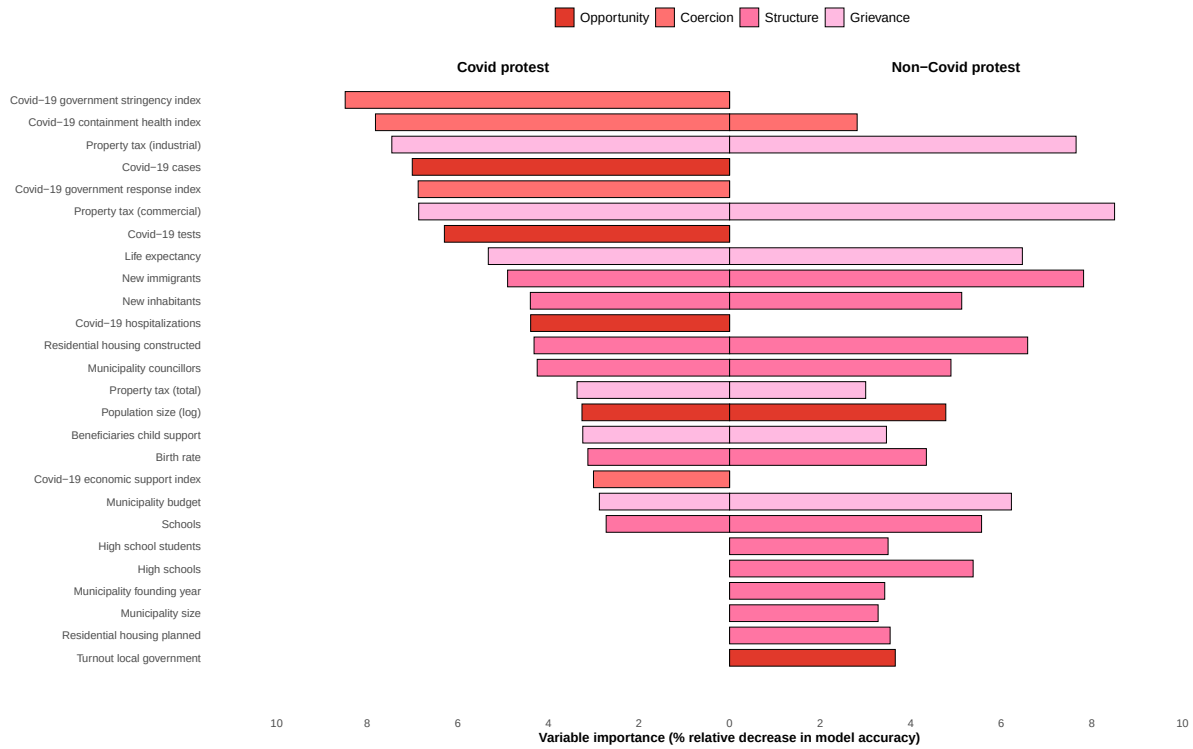

Figure 7: Top-20 important drivers of protest, per type based on the binary random forest classification model (Figure S12).

```
"Arab population",
"Jewish orthodox majority",
"Religion majority",
"Religion (%)",
"Property tax (total)",
"Property tax (residential)" ,
"Property tax (commercial)",
"Property tax (industrial)",
"Municipality budget",
"Socio-economic index",
"Socio-economic cluster",
"Unemployment daily rate",
"Unemployment rate",
"Beneficiaries child support",
"Life expectancy",
"Divorce rate",
"Dependence ratio",
"Birth rate",
"Aged +65",
"Aged <18",
"Student rate",
"Drop-out rate",
"University degree rate",
"Unemployment mean age",
"Population growth rate",
```

```
                  "Fertility rate",
                  "New inhabitants",
                  "New immigrants",
                  "Immigrant rate",
                  "Residential housing planned",
                  "Residential housing constructed",
                  "Municipality founding year",
                  "Municipality councillors",
                  "High schools",
                  "Schools",
                  "Paid teacher rate",
                  "Children per teacher",
                  "High school students",
                  "Diabetes rate",
                  "Overweight rate",
                  "Municipality size",
                  "Municipality border beach",
                  "Distance to Tel Aviv",
                  "Peripherality index",
                  "Population density",
                  "Population size (log)",
                  "Covid-19 cases",
                  "Covid-19 hospitalizations",
                  "Covid-19 tests",
                  "Covid-19 deaths",
                  "Turnout local government",
                  "Turnout Knesset 2019",
                  "Turnout Knesset 2020",
                  "Turnout Knesset 2021",
                  "Covid-19 government stringency index",
                  "Covid-19 economic support index",
                  "Covid-19 containment health index",
                  "Covid-19 government response index")

# remove columns (temporarily) where we will not impute/standardize
df <- df_save[,c(-1:-5)]
pro <- df_save[,c(3:5)]

# impute and standardize covariates
pre.model <- preProcess(df, method= c("knnImpute", "center", "scale"))
df <- predict(pre.model, newdata = df)
df <- data.frame(df)

# get protest counts back into the data frame
df <- cbind(pro, df)

# save new df data to import
df_save <- cbind(df_save[,1:2],df)

# pull out time column
week <- df_save$week_id

# pull out locality column for per locality partition
```

```
locality <- as.numeric(df_save$locality_id)
```

6.1 Poisson regression for count outcomes

For the count outcome, we first use a spatial split to look at Covid protest (Section 6.1.1), then non-Covid protest outcomes (Section 6.1.2).

6.1.1 Covid protest with spatial split

```
# subset to Covid protest
df <- df_save
df <- df[, c(-1:-3, -5)]

# rename protest variable
colnames(df)[1] = "protest"

# locality-based split (80/20)
# splits the data into training and testing sets, ensuring that all
# observations from a local government are grouped into the same subset
train.rows <- CreateLocalityPartition(df[, 1], locality = locality, p=0.8)
df.train <- df[train.rows,]
df.test <- df[-train.rows,]

# pull out protest from training and test data
y_train <- df.train[, 1]
y_test <- df.test[, 1]

# define number of folds on protest
folds.10 <- createFolds(y_train, k = 10)

# define trainControl parameters
control <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 5,
  verboseIter = TRUE,
  savePredictions = "final",
  index = folds.10
)

# run count model from poisson family of glm
model_glm <- train(
  protest ~ .,
  data = df.train,
  method = "glm",
  family = "poisson",
  trControl = control
)
model_glm_covid.count_full <- model_glm

# optional: save best model as .Rda
#save(model_glm_covid.count_full,
#     file="results/255_model_glm_covid.count_full.Rda")
```

```r
# in-sample prediction: model fit
glm.fitted <- predict(model_glm)

# out-of-sample prediction: model performance
glm.predicted <- predict(model_glm, df.test)

# calculate performance statistics ("in" and "out" of sample)
performance <- list(
  RMSE.in = rmse(df.train$protest, glm.fitted),
  cor.in = cor(df.train$protest, glm.fitted),
  R2.in = cor(df.train$protest, glm.fitted)^2,
  MAE.in = mae(df.train$protest, glm.fitted),
  RMSE.out = rmse(df.test$protest, glm.predicted),
  cor.out = cor(df.test$protest, glm.predicted),
  R2.out = cor(df.test$protest, glm.predicted)^2,
  MAE.out = mae(df.test$protest, glm.predicted))

# save performance statistics
performance_glm_covid.count_full_255 <- performance
save(performance_glm_covid.count_full_255,
     file = "results/255_performance_glm_covid.count_full.Rda")
```

6.1.2 Non-Covid protest with spatial split

```r
# subset to non-Covid protest
df <- df_save
df <- df[, c(-1:-4)]

# rename protest variable
colnames(df)[1] = "protest"

# locality-based split (80/20)
# splits the data into training and testing sets, ensuring that all
# observations from a local government are grouped into the same subset
train.rows <- CreateLocalityPartition(df[, 1], locality = locality, p=0.8)
df.train <- df[train.rows,]
df.test <- df[-train.rows,]

# pull out protest from training and test data
y_train <- df.train[, 1]
y_test <- df.test[, 1]

# define number of folds on protest
folds.10 <- createFolds(y_train, k = 10)

# define trainControl parameters
control <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 5,
  verboseIter = TRUE,
  savePredictions = "final",
  index = folds.10
```

```
)

# run count model from poisson family of glm
model_glm <- train(
  protest ~ .,
  data = df.train,
  method = "glm",
  family = "poisson",
  trControl = control
)
model_glm_non_covid.count_full <- model_glm

# optional: save best model as .Rda
#save(model_glm_non_covid.count_full,
#     file="results/255_model_glm_non_covid.count_full.Rda")

# in-sample prediction: model fit
glm.fitted <- predict(model_glm)

# out-of-sample prediction: model performance
glm.predicted <- predict(model_glm, df.test)

# calculate performance statistics ("in" and "out" of sample)
performance <- list(
  RMSE.in = rmse(df.train$protest, glm.fitted),
  cor.in = cor(df.train$protest, glm.fitted),
  R2.in = cor(df.train$protest, glm.fitted)^2,
  MAE.in = mae(df.train$protest, glm.fitted),
  RMSE.out = rmse(df.test$protest, glm.predicted),
  cor.out = cor(df.test$protest, glm.predicted),
  R2.out = cor(df.test$protest, glm.predicted)^2,
  MAE.out = mae(df.test$protest, glm.predicted))

# save performance statistics
performance_glm_non_covid.count_full_255 <- performance
save(performance_glm_non_covid.count_full_255,
     file = "results/255_performance_glm_non_covid.count_full.Rda")
```

6.2 Random forest regression

For the count outcome, we first use a spatial split to look at Covid protest (Section 6.2.1), then non-Covid protest outcomes (Section 6.2.2).

6.2.1 Covid protest with spatial split

```
# subset to Covid protest
df <- df_save
df <- df[, c(-1:-3, -5)]

# rename protest variable
colnames(df)[1] = "protest"

# one-hot encoding of categorical variables
```

```
oneHot.model <- dummyVars(" ~ .", data=df)
df <- predict(oneHot.model, newdata = df)
df <- data.frame(df)

# locality-based split (80/20)
# splits the data into training and testing sets, ensuring that all
# observations from a local government are grouped into the same subset
train.rows <- CreateLocalityPartition(df[, 1], locality = locality, p=0.8)
df.train <- df[train.rows,]
df.test <- df[-train.rows,]

# pull out protest from training and test data
y_train <- df.train[, 1]
y_test <- df.test[, 1]

# define number of folds on protest
folds.10 <- createFolds(y_train, k = 10)

# specify 10 folds and repeat 5 times
control <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 5,
  verboseIter = TRUE,
  savePredictions = "final",
  search="grid",
  index = folds.10)

# create tune grid with values from 3:12 for mtry
tunegrid <- expand.grid(.mtry=c(3:12))

# define ntrees, i.e. the number of trees to grow
ntrees <- c(50,80,100,200,500,1000,1500,2000)

# run grid search
modellist <- list()
for (ntree in ntrees) {
  fit = train(
    protest ~ .,
    tuneLength = 5,
    importance = TRUE,
    data = df.train,
    method = "rf",
    trControl = control,
    tuneGrid = tunegrid,
    ntree = ntree)
  key <- toString(ntree)
  modellist[[key]] <- fit
}
# optional: save scan configuration
#save(modellist,file="rf_optimization/255_covid.count_full_modellist.Rda")

# compare results
```

```r
results <- resamples(modellist)
summary(results)
pdf(file = "figures/255_rf_covid.count_full.pdf", width = 6, height = 3.5)
dotplot(results)
dev.off()

# select best model based on smallest RMSE/MAE and greatest Rsquared
ntree_opt <- "1500"
model_rf <- modellist[[ntree_opt]]
mtry_opt <- as.numeric(unlist(model_rf$bestTune))

# optional: save best model as .Rda
#model_rf_covid.count_full <- modellist[[ntree_opt]]
#save(model_rf_covid.count_full,
#     file="results/255_model_rf_covid.count_full.Rda")

# in-sample prediction: model fit
rf.fitted <- predict(model_rf)

# out-of-sample prediction: model performance
rf.predicted <- predict(model_rf, df.test)

# calculate performance statistics ("in" and "out" of sample)
performance <- list(
  RMSE.in = rmse(df.train$protest, rf.fitted),
  cor.in = cor(df.train$protest, rf.fitted),
  R2.in = cor(df.train$protest, rf.fitted)^2,
  MAE.in = mae(df.train$protest, rf.fitted),
  RMSE.out = rmse(df.test$protest, rf.predicted),
  cor.out = cor(df.test$protest, rf.predicted),
  R2.out = cor(df.test$protest, rf.predicted)^2,
  MAE.out = mae(df.test$protest, rf.predicted))

# save performance statistics
performance_rf_covid.count_full_255 <- performance
save(performance_rf_covid.count_full_255,
     file = "results/255_performance_rf_covid.count_full.Rda")


# variable permutation importance

# rerun classification using cforest from party pkg
rf.corr <- party::cforest(
  protest ~ .,
  data = df.train,
  control = cforest_unbiased(mtry = mtry_opt, ntree = as.numeric(ntree_opt))
)

# calculate unconditional variable importance
# (using faster implementation of permimp pkg)
varImp <- permimp::permimp(rf.corr, conditional = FALSE, asParty = TRUE)
varImp <- sort(varImp$values, decreasing = TRUE)
```

```r
# get top-20 variables
top20 <- data.frame(varImp[c(1:20)])
top20$names <- row.names(top20)
row.names(top20) <- NULL
names(top20) <- c("%IncMSE", "names")
top20.vec <- top20$names

# save top-20 variables as .csv (used in appendix figure)
write.csv(top20,"results/255_top20.rf_covid.count_full.csv",
          row.names = FALSE)


# minimal model

# subset train and test data to top 20 identified by full model
df.train <- df.train[,names(df.train) %in% top20.vec]
df.test <- df.test[,names(df.test) %in% top20.vec]

# add protest to training and test data
df.train$protest <- y_train
df.test$protest <- y_test

# run grid search
modellist <- list()
for (ntree in ntrees) {
  fit = train(
    protest ~ .,
    tuneLength = 5,
    importance = TRUE,
    data = df.train,
    method = "rf",
    trControl = control,
    tuneGrid = tunegrid,
    ntree = ntree)
  key <- toString(ntree)
  modellist[[key]] <- fit
}
# optional: save scan configuration
#save(modellist,file="rf_optimization/255_covid.count_min_modellist.Rda")

# compare results
results <- resamples(modellist)
summary(results)
pdf(file = "figures/255_rf_covid.count_min.pdf", width = 6, height = 3.5)
dotplot(results)
dev.off()

# select best model based on smallest RMSE/MAE and greatest Rsquared
ntree_opt <- "200"
model_rf <- modellist[[ntree_opt]]

# optional: save best model as .Rda
#model_rf_covid.count_min <- modellist[[ntree_opt]]
```

```
#save(model_rf_covid.count_min,
#     file="results/255_model_rf_covid.count_min.Rda")

# in-sample prediction: model fit
rf.fitted <- predict(model_rf)

# out-of-sample prediction: model performance
rf.predicted <- predict(model_rf, df.test)

# calculate performance statistics ("in" and "out" of sample)
performance <- list(
  RMSE.in = rmse(df.train$protest, rf.fitted),
  cor.in = cor(df.train$protest, rf.fitted),
  R2.in = cor(df.train$protest, rf.fitted)^2,
  MAE.in = mae(df.train$protest, rf.fitted),
  RMSE.out = rmse(df.test$protest, rf.predicted),
  cor.out = cor(df.test$protest, rf.predicted),
  R2.out = cor(df.test$protest, rf.predicted)^2,
  MAE.out = mae(df.test$protest, rf.predicted))

# save performance statistics
performance_rf_covid.count_min_255 <- performance
save(performance_rf_covid.count_min_255,
     file = "results/255_performance_rf_covid.count_min.Rda")
```

6.2.2 Non-Covid protest with spatial split

```
# subset to non-Covid protest
df <- df_save
df <- df[, c(-1:-4)]

# rename protest variable
colnames(df)[1] = "protest"

# one-hot encoding of categorical variables
oneHot.model <- dummyVars(" ~ .", data=df)
df <- predict(oneHot.model, newdata = df)
df <- data.frame(df)

# locality-based split (80/20)
# splits the data into training and testing sets, ensuring that all
# observations from a local government are grouped into the same subset
train.rows <- CreateLocalityPartition(df[, 1], locality = locality, p=0.8)
df.train <- df[train.rows,]
df.test <- df[-train.rows,]

# pull out protest from training and test data
y_train <- df.train[, 1]
y_test <- df.test[, 1]

# define number of folds on protest
folds.10 <- createFolds(y_train, k = 10)
```

```r
# specify 10 folds and repeat 5 times
control <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 5,
  verboseIter = TRUE,
  savePredictions = "final",
  search="grid",
  index = folds.10)

# create tune grid with values from 3:12 for mtry
tunegrid <- expand.grid(.mtry=c(3:12))

# define ntrees, i.e. the number of trees to grow
ntrees <- c(50,80,100,200,500,1000,1500,2000)

# run grid search
modellist <- list()
for (ntree in ntrees) {
  fit = train(
    protest ~ .,
    tuneLength = 5,
    importance = TRUE,
    data = df.train,
    method = "rf",
    trControl = control,
    tuneGrid = tunegrid,
    ntree = ntree)
  key <- toString(ntree)
  modellist[[key]] <- fit
}
# optional: save scan configuration
#save(modellist,file="rf_optimization/255_non_covid.count_full_modellist.Rda")

# compare results
results <- resamples(modellist)
summary(results)
pdf(file = "figures/255_rf_non_covid.count_full.pdf", width = 6, height = 3.5)
dotplot(results)
dev.off()

# select best model based on smallest RMSE/MAE and greatest Rsquared
ntree_opt <- "2000"
model_rf <- modellist[[ntree_opt]]
mtry_opt <- as.numeric(unlist(model_rf$bestTune))

# optional: save best model as .Rda
#model_rf_non_covid.count_full <- modellist[[ntree_opt]]
#save(model_rf_non_covid.count_full,
#     file="results/255_model_rf_non_covid.count_full.Rda")

# in-sample prediction: model fit
rf.fitted <- predict(model_rf)
```

```r
# out-of-sample prediction: model performance
rf.predicted <- predict(model_rf, df.test)

# calculate performance statistics ("in" and "out" of sample)
performance <- list(
  RMSE.in = rmse(df.train$protest, rf.fitted),
  cor.in = cor(df.train$protest, rf.fitted),
  R2.in = cor(df.train$protest, rf.fitted)^2,
  MAE.in = mae(df.train$protest, rf.fitted),
  RMSE.out = rmse(df.test$protest, rf.predicted),
  cor.out = cor(df.test$protest, rf.predicted),
  R2.out = cor(df.test$protest, rf.predicted)^2,
  MAE.out = mae(df.test$protest, rf.predicted))

# save performance statistics
performance_rf_non_covid.count_full_255 <- performance
save(performance_rf_non_covid.count_full_255,
     file = "results/255_performance_rf_non_covid.count_full.Rda")


# variable permutation importance

# rerun classification using cforest from party pkg
rf.corr <- party::cforest(
  protest ~ .,
  data = df.train,
  control = cforest_unbiased(mtry = mtry_opt, ntree = as.numeric(ntree_opt))
)

# calculate unconditional variable importance
# (using faster implementation of permimp pkg)
varImp <- permimp::permimp(rf.corr, conditional = FALSE, asParty = TRUE)
varImp <- sort(varImp$values, decreasing = TRUE)

# get top-20 variables
top20 <- data.frame(varImp[c(1:20)])
top20$names <- row.names(top20)
row.names(top20) <- NULL
names(top20) <- c("%IncMSE", "names")
top20.vec <- top20$names

# save top-20 variables as .csv (used in appendix figure)
write.csv(top20,"results/255_top20.rf_non_covid.count_full.csv",
          row.names = FALSE)


# minimal model

# subset train and test data to top 20 identified by full model
df.train <- df.train[,names(df.train) %in% top20.vec]
df.test <- df.test[,names(df.test) %in% top20.vec]

# add protest to training and test data
```

```
df.train$protest <- y_train
df.test$protest <- y_test

# run grid search
modellist <- list()
for (ntree in ntrees) {
  fit = train(
    protest ~ .,
    tuneLength = 5,
    importance = TRUE,
    data = df.train,
    method = "rf",
    trControl = control,
    tuneGrid = tunegrid,
    ntree = ntree)
  key <- toString(ntree)
  modellist[[key]] <- fit
}
# optional: save scan configuration
#save(modellist,file="rf_optimization/255_non_covid.count_min_modellist.Rda")

# compare results
results <- resamples(modellist)
summary(results)
pdf(file = "figures/255_rf_non_covid.count_min.pdf", width = 6, height = 3.5)
dotplot(results)
dev.off()

# select best model based on smallest RMSE/MAE and greatest Rsquared
ntree_opt <- "80"
model_rf <- modellist[[ntree_opt]]

# optional: save best model as .Rda
#model_rf_covid.count_min <- modellist[[ntree_opt]]
#save(model_rf_covid.count_min,
#     file="results/255_model_rf_non_covid.count_min.Rda")

# in-sample prediction: model fit
rf.fitted <- predict(model_rf)

# out-of-sample prediction: model performance
rf.predicted <- predict(model_rf, df.test)

# calculate performance statistics ("in" and "out" of sample)
performance <- list(
  RMSE.in = rmse(df.train$protest, rf.fitted),
  cor.in = cor(df.train$protest, rf.fitted),
  R2.in = cor(df.train$protest, rf.fitted)^2,
  MAE.in = mae(df.train$protest, rf.fitted),
  RMSE.out = rmse(df.test$protest, rf.predicted),
  cor.out = cor(df.test$protest, rf.predicted),
  R2.out = cor(df.test$protest, rf.predicted)^2,
  MAE.out = mae(df.test$protest, rf.predicted))
```

```r
# save performance statistics
performance_rf_non_covid.count_min_255 <- performance
save(performance_rf_non_covid.count_min_255,
     file = "results/255_performance_rf_non_covid.count_min.Rda")
```

Figure S15

Figure S15 illustrates the top-20 important drivers of protest using the locality-based split for the full sample of 255 local governments, per type and based on the full random forest model.

```r
# import (sorted) top-20 Covid protest drivers
drivers.cov <- read.csv("results/255_top20.rf_covid.count_full.csv")
colnames(drivers.cov) <- c("value", "names")

# import (sorted) top-20 non-Covid protest drivers
drivers.noncov <- read.csv("results/255_top20.rf_non_covid.count_full.csv")
colnames(drivers.noncov) <- c("value", "names")

# convert to relative importance values (in %)
drivers.cov.total <- sum(drivers.cov$value)
drivers.cov$value <- -100*drivers.cov$value/drivers.cov.total
drivers.noncov.total <- sum(drivers.noncov$value)
drivers.noncov$value <- 100*drivers.noncov$value/drivers.noncov.total

# add missing rows from one df in the other
drivers <- merge(drivers.cov, drivers.noncov, by= c("names"),
                 all.x = TRUE, all.y = TRUE)

drivers.covid <- drivers[, c(1,2)]
colnames(drivers.covid)[2] ="value"
drivers.covid

drivers.noncov <- drivers[, c(1,3)]
colnames(drivers.noncov)[2] ="value"
drivers.noncov

# add column with protest type
drivers.covid$type <- "Covid protests"
drivers.noncov$type <- "Non-Covid protests"

# replace NA with 0
drivers.covid[is.na(drivers.covid$value),]$value <- 0
drivers.noncov[is.na(drivers.noncov$value),]$value <- 0

# rbind
df.plot <- rbind(drivers.covid, drivers.noncov)

# import list with variables and mechanisms
df.mechanism <- read.csv("data/raw_mechanism.csv")

# get mechanisms and variable names
df.plot <- merge(df.plot, df.mechanism, by="names", all.x=TRUE)
df.plot$type <- as.factor(df.plot$type)
```

```r
# indicator levels
df.plot <- df.plot[order(df.plot$type, df.plot$value),]
lvls <- df.plot$indicator
lvls <- unique(lvls)
df.plot$indicator <- factor(df.plot$indicator, levels=c(lvls))

# mechanism levels
lvls.m <- c("Opportunity", "Coercion", "Structure", "Grievance")
df.plot$mechanism <- factor(df.plot$mechanism, levels=c(lvls.m))
df.plot <- df.plot[!is.na(df.plot$indicator),]

##### plot

fig_s15 <- ggplot2::ggplot() +
  geom_bar(data = df.plot,
           aes(x=value, y=fct_rev(indicator), fill= mechanism),
           stat = "identity", colour="black", linewidth= .1,
           width = .75, alpha = 0.9, na.rm = TRUE) +
  scale_fill_manual(name = "",
                    values = c("#de2414","#ff6161",
                               "#ff6699","#ffb3de"),
                    labels = c("Opportunity", "Coercion",
                               "Structure","Grievance"),
                    na.translate = F) +
  annotate("text", x = c(-5, 5), y = c(29.5,29.5),
           label = c("Covid protest", "Non-Covid protest"),
           size = 4.5, fontface = "bold") +
  ggplot2::labs(
    y = " ",
    x = "Variable importance (% relative decrease in model accuracy)") +
  scale_y_discrete(expand=c(0.1,0)) +
  scale_x_continuous(limits = c(-10.5, 10.5),
                     breaks=c(-10, -8, -6, -4, -2, 0,
                              2, 4, 6, 8, 10),
                     labels=c("10","8","6","4","2","0",
                             "2","4","6","8","10")) +
  theme_minimal() +
  theme(panel.grid.major.x = element_blank(),
        panel.grid.minor.x = element_blank(),
        panel.grid.minor.y = element_blank(),
        panel.grid.major.y = element_blank(),
        axis.text.x = element_text(size =10),
        axis.title.x  = element_text(size =12, face = "bold"),
        axis.text.y =  element_text(size =10),
        axis.title.y = element_text(size=12),
        legend.position= c(.5, 1.04),
        legend.direction = "horizontal",
        legend.text = element_text(size=12),
        plot.title = element_blank(),
        plot.subtitle = element_blank(),
        plot.background = element_blank(),
        plot.margin=unit(c(4, 0.5, 2, 0.5), units="line")) #t, r, b, l)
```

```
fig_s15
```

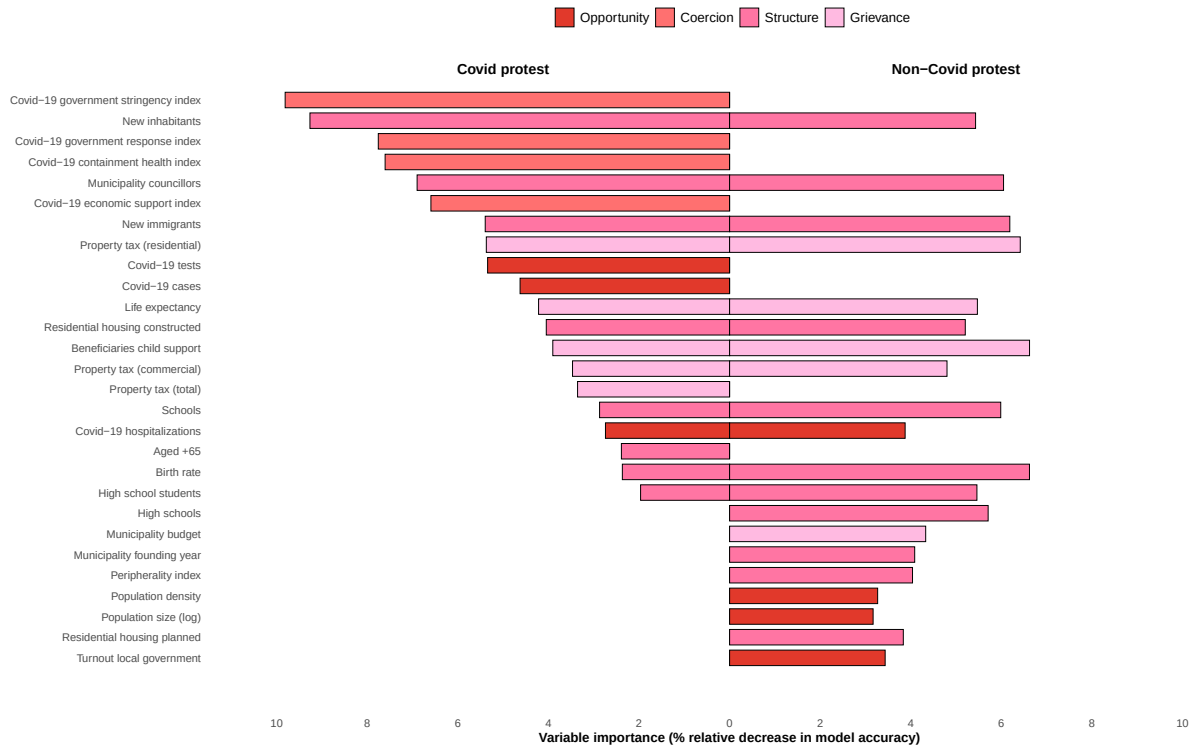

Figure 8: Alternative sample: Top-20 important drivers of protest, per type and based on the full random forest model (Figure S15).

```
# optional: save Figure S15 as .svg or .pdf
#ggsave("figures/si_fig_s15.svg", fig_s15,
#       width=16.79, height=10.65, units="in", dpi=300)
#ggsave("figures/si_fig_s15.pdf", fig_s15, width=16.79,
#       height=10.65, units="in", dpi=300)
```