Multinomial CI Prediction for 2020 Election

Shanghao Zhong

2020-11-06

Load package

```
library(jsonlite)
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
library(MultinomialCI)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

You also need to load the data. You should have the `all.state.dataset` ready when data is load correctly.

```
load("MultinomialCIProject2020.RData")
all.state.dataset <- get('all.state.dataset')
summary(all.state.dataset)
```

```
##              Length Class  Mode
## arizona       8     -none- list
## pennsylvania 13     -none- list
## nevada        5     -none- list
## georgia       9     -none- list
## alaska        1     -none- list
```

Predifine function

The script concerns about the probability of a vote cast for a specific candidate. Assuming the votes follow a multinomial distribution, it uses the `MultinomialCI` package to calculate the confidence interval of the probability for each candidate.

```r
#' Calculate county level multinomial c.i.
#'
#' @param df a data frame with vote counts.
#'           row.name is county name,
#'           and each column is the number of votes for each candidate.
#' @param alpha The significance level for the confidence intervals.
#'              Must be a real number in the interval [0, 1]
#' @return a data frame of confidence intervals for each candidate
#' @export
CI.by.county <- function(df, alpha) {
  df <- filter(df, df[[1]] > 0)
  candidate.names <- names(df)
  county <- row.names(df)
  mat <- t(apply(df, 1, as.numeric))

  CI.per.county <- function(vec) {
    CI.per.county <- c(multinomialCI(pmax(vec, 0), alpha))
  }
  cols <- apply(expand.grid(candidate.names, c('low', 'high')), 1, paste, collapse='.')
  all.ci <- t(apply(mat, 1, CI.per.county))

  output <- data.frame(all.ci, row.names = county)
  colnames(output) <- cols

  return(output)
}
```

With the probability of a vote cast for a candidate, the final projection is done by $np \pm z \cdot \sqrt{n \cdot p \cdot (1-p)}$, where p is the probability of a candidate winning a vote, n is the expected remaining vote according to NYT.

```r
#' Calculate prediction interval of the remaining votes given the
#' probability.
#'
#' @param remaining A scalar or vector containing the remaining votes in the
#'                  county.
#' @param prob A scalar or vector containing the probability of votes going to
#'             the candidate in the county.
#' @param alpha confidence level
#'
#' @return a length 2 vector, first is the lower bound and the second is the
#'         upper.
#' @export
p.interval <- function(remaining, prob, alpha=0.95) {
  lo <- remaining*prob + qnorm((1-alpha)/2)*sqrt(remaining*prob*(1-prob))
  hi <- remaining*prob - qnorm((1-alpha)/2)*sqrt(remaining*prob*(1-prob))
  return(c(lo = sum(lo, na.rm = TRUE), hi = sum(hi, na.rm = TRUE)))
}
```

```r
#' Helper function to convert UTC time string to EST
#'
#' @param timestring string representation of time, NYT's time in ISO format
#'
#' @return a POSIXct object in EST
#' @export
#'
```

```r
#' @examples to_EST('2020-11-07T01:46:10Z')
to.EST <- function(timestring) {
  with_tz(parse_date_time(timestring, 'ymd HMS'), 'EST')
}
```

More helper functions to pull data from NYT.

- `collect.data(dataset, state)` pulls data from NYT.
- `get.state.data(dataset, state, index)` gets historic data of a state.
- `load.state.data(state.data, names)` saves state data into global environment.

```r
#' Collect data from NYT of the given state and return the updated dataset.
#'
#' @param dataset The dataset that store all previously pull data.
#'                It should be a list of all tracked states,
#'                  each is another list that contains historical snapshot.
#' @param state What state we want to get the data for.
#'              White spaces should be replaced by `-`.
#'              For example, New York should be `new-york`,
#'                Pennsylvania should be `pennsylvania`, and
#'                D.C. should be `district-of-columbia`.
#'
#' @return The updated dataset, in which the new data is appended at the end of
#'           its state list.
#' @export
#'
#' @examples
#' # this will read arizona data, update the dataset, and assign it back.
#' all.state.dataset %>% collect.data('arizona') -> all.state.dataset
collect.data <- function(dataset, state) {
  nyt.api <- paste(
    'https://static01.nyt.com/elections-assets/2020/data/api/2020-11-03/race-page/',
    state,
    '/president.json',
    sep = '')

  results <- fromJSON(nyt.api)
  current.time <- max(results$data$races$counties[[1]]$last_updated)

  if (!state %in% names(dataset)) {
    update.type <- "New state"
    dataset[[state]] <- list()
    dataset[[state]][[current.time]] <- results$data$races$counties[[1]]
  } else { # if the state has been track, see if this one is new update
    previous.time <- max(last(dataset[[state]])$last_updated)
    if (current.time == previous.time) {
      update.type <- "No update"
    } else {
      update.type <- "New update"
      dataset[[state]][[current.time]] <- results$data$races$counties[[1]]
    }
  }

  cat(paste(state, ":", update.type),
      paste("Update time:", to.EST(current.time)),
      paste("Current margin for Biden:",
            sum(last(dataset[[state]])$results$bidenj)
            - sum(last(dataset[[state]])$results$trumpd)),
      sep='\n')

  return(dataset)
}


#' Get a state's data of a historic snapshot.
#'
#' @param dataset The dataset that stores all state's data
#' @param state The state of which you are getting the data
#' @param index The index of the historical snapshot of this state.
#'              The latest snapshot will be pulled by default.
#'
#' @return
#' a list containing 3 elements:
#' - `state.details`: all the details of this snapshot, by county,
#' - `all.votes`: all vote counts of each candidate, by county, and
#' - `mail.votes`: mail vote counts of each candidate, by county.
#'
#' @export
#'
#' @examples
#' all.state.dataset %>% get.state.data('arizona')
get.state.data <- function(dataset, state, index = NULL) {
  state.data <- dataset[[state]]
  if (is.null(index)) {
    index <- length(state.data)
  }
  state.details <- state.data[[index]]

  cat('Get state data for', state,
      paste('(updated at: ', to.EST(max(state.details$last_updated)), ').\n',
            sep = ''))

  return(list(
    state.details = state.details,
    all.votes = data.frame(state.details$results,
                            row.names = state.details$name),
    mail.votes = data.frame(state.details$results_absentee,
                             row.names = state.details$name)
  ))
}

#' Load state data into global environment.
#'
#' @param state.data The state data, usually the return from `get.state.data`
#' @param names The global variable names to store `state.details`, `all.votes`,
#'                  and `mail.votes` from `state.data`
#'
#' @return NULL
```

```r
#' @export
#'
#' @examples
#' # load latest data
#' get.state.data(all.state.dataset, current.state.name) %>% load.state.data()
#' # load first snapshot
#' get.state.data(all.state.dataset, current.state.name, 1) %>%
#'   load.state.data(c('old.state.details', 'old.all.votes', 'old.mail.votes'))
load.state.data <- function(state.data, names = c('state.details',
                                                  'all.votes',
                                                  'mail.votes')) {
  if(length(names) != 3 & typeof(names) != 'character') {
    stop('names should be a character vector with 3 elements.')
  }
  assign(names[1], state.data$state.details, envir = .GlobalEnv, inherits = TRUE)
  assign(names[2], state.data$all.votes, envir = .GlobalEnv, inherits = TRUE)
  assign(names[3], state.data$mail.votes, envir = .GlobalEnv, inherits = TRUE)
  cat('State data are loaded in: ',
      paste(names, collapse = ', '),
      '.\n',
      sep = '')
}
```

Pull data

Pull new data from a state from NYT's API. `current.state.name` needs to be the fully spelled name in lowercase in which spaces is replaced by `-`. For example, New York will be `new-york`; Pennsylvania will be `pennsylvania`, and D.C. will be `district-of-columbia`.

```r
current.state.name <- 'arizona'
all.state.dataset %>% collect.data(current.state.name) -> all.state.dataset
```

```
## arizona : No update
## Update time: 2020-11-07 20:02:36
## Current margin for Biden: 18713
```

```r
all.state.dataset %>% get.state.data(current.state.name) %>% load.state.data()
```

```
## Get state data for arizona (updated at: 2020-11-07 20:02:36).
## State data are loaded in: state.details, all.votes, mail.votes.
```

See the number of snapshots saved

```r
sapply(all.state.dataset, length)
```

```
##      arizona pennsylvania       nevada      georgia       alaska
##            8           13            5            9            1
```

See the time at which snapshots were taken (in EST)

```r
to.EST(names(all.state.dataset[[current.state.name]]))
```

```
## [1] "2020-11-06 11:09:33 EST" "2020-11-06 15:27:34 EST"
## [3] "2020-11-06 20:25:37 EST" "2020-11-06 21:01:47 EST"
## [5] "2020-11-06 21:04:35 EST" "2020-11-07 11:01:39 EST"
## [7] "2020-11-07 16:27:38 EST" "2020-11-07 20:02:36 EST"
```

Estimate the probability for each candidate

You can change `using` to update how you want to estimate the probability. Unhide one of them and hide to other to use.

first option:

- Use the difference between old data and new data
- Best for predicting the most recent trend
- Doesn't work if the difference between old data and new data is small or non-representative

second option:

- Use the mail.votes to predict old data and new data
- Work the best if the mail votes is homogeneous throughout different time
- Doesn't work But the demographics within mail data can change over time

When `using` is incomplete, the probability of each candidate will based on `all.votes`

```
# first option
using <- data.frame(
  data.matrix(get.state.data(all.state.dataset, current.state.name)$all.votes)
  - data.matrix(get.state.data(all.state.dataset, current.state.name, 1)$all.votes),
  row.names = get.state.data(all.state.dataset, current.state.name)$state.details$name)
```

```
## Get state data for arizona (updated at: 2020-11-07 20:02:36).
## Get state data for arizona (updated at: 2020-11-07 20:02:36).
## Get state data for arizona (updated at: 2020-11-06 11:09:33).
```

```
# second option
# using <- get.state.data(all.state.dataset, current.state.name)$mail.votes

remaining <- data.frame(exp.remaining = pmax(0, state.details$tot_exp_vote - rowSums(data.matrix(all.vo
                        row.names = state.details$name)
ci.mail <- merge(remaining, CI.by.county(using, 0.95), by=0)
ci.other <- merge(remaining, CI.by.county(all.votes, 0.95), by=0)
ci.other <- ci.other[ci.other$Row.names %in%
                       setdiff(ci.other$Row.names, ci.mail$Row.names), ]
est <- rbind(ci.mail, ci.other)
rm(remaining)

est <- est[order(est$Row.names), ]
rownames(est) <- est$Row.names
est$Row.names <- NULL
est[est$exp.remaining > 0, ]
```

```
##            exp.remaining bidenj.low trumpd.low jorgensenj.low write.ins.low
## Apache              8645  0.8228963  0.1634051     0.01076321  0.0000000000
## Cochise             6619  0.3402394  0.6233886     0.03222836  0.0000000000
## Coconino             587  0.4404372  0.5256831     0.02732240  0.0000000000
## La Paz               323  0.3062753  0.6794968     0.01108282  0.0001497679
## Maricopa           43567  0.3999057  0.5133226     0.02690449  0.0582328213
## Mohave              2164  0.1743745  0.8003538     0.02223907  0.0000000000
## Navajo              5536  0.3179027  0.6565895     0.02125650  0.0000000000
## Pima               25936  0.4366584  0.4743104     0.02810825  0.0546476664
## Pinal              19386  0.3538980  0.6213667     0.01910491  0.0031280077
## Santa Cruz          1454  0.6707766  0.3156144     0.01089737  0.0006650977
## Yuma                2105  0.4939357  0.4768041     0.02607641  0.0000000000
##            bidenj.high trumpd.high jorgensenj.high write.ins.high
## Apache       0.8259155   0.1664243      0.01378238    0.002040700
## Cochise      0.3438750   0.6270242      0.03586394    0.002254368
## Coconino     0.4455235   0.5307694      0.03240879    0.002900593
## La Paz       0.3080035   0.6812250      0.01281100    0.001877950
## Maricopa     0.4007252   0.5141421      0.02772403    0.059052360
## Mohave       0.1766075   0.8025868      0.02447208    0.001222145
## Navajo       0.3211672   0.6598541      0.02452105    0.001847452
## Pima         0.4399260   0.4775780      0.03137591    0.057915324
## Pinal        0.3551755   0.6226442      0.02038239    0.004405485
## Santa Cruz   0.6718347   0.3166725      0.01195543    0.001723160
## Yuma         0.4960816   0.4789500      0.02822226    0.001084606
```

Estimate the final range

With the lower and upper bound of Biden’s and Trump’s probability in a county, we use both probability to calculate the CI. The lower end of the CI from the low probability is a candidate’s lower bound, while the upper end of the CI from the high probility is a candidate’s upper bound. We then see the margin using Biden’s lower bound − Trump’s upper bound, and using Biden’s upper bound − Trump’s lower bound, to calculate the final projection interval.

```
future.lo <- p.interval(est$exp.remaining, est$bidenj.low)[1] - p.interval(est$exp.remaining, est$trumpd
future.hi <- p.interval(est$exp.remaining, est$bidenj.high)[2] - p.interval(est$exp.remaining, est$trump
current.diff <- sum(all.votes$bidenj) - sum(all.votes$trumpd)
current.diff + c(future.lo, future.hi)
```

```
##        lo        hi
##  6642.074 10537.350
```

Saving the data

```
save.image("MultinomialCIProject2020.RData")
```