Multinomial CI Prediction for 2020 Election

Shanghao Zhong

2020-11-06

Required package and `RData`

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
```

```
library(purrr)
```

```
##
## Attaching package: 'purrr'
```

```
## The following object is masked from 'package:jsonlite':
##
##     flatten
```

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

Note: you might need to upgrade R to load this `RData` correctly.

```
load("MultinomialCIProject2020.RData")
# set thsi to `list()` if you want to start fresh
all.state.dataset <- get('all.state.dataset')
summary(all.state.dataset)
```

```
##             Length Class  Mode
## arizona     12     -none- list
```

```
## pennsylvania 16     -none- list
## nevada        5     -none- list
## georgia      13     -none- list
## alaska        1     -none- list
```

Predifine function

Confidence interval and prediction interval calculation

The model 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.

```
#' 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=0.05) {
  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.

```
#' 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.
```

```r
#' @export
p.interval <- function(remaining, prob, alpha=0.05) {
  lo <- remaining*prob + qnorm((alpha)/2)*sqrt(remaining*prob*(1-prob))
  hi <- remaining*prob - qnorm((alpha)/2)*sqrt(remaining*prob*(1-prob))
  return(c(lo = sum(lo, na.rm = TRUE), hi = sum(hi, na.rm = TRUE)))
}
```

Helper functions to pull data from NYT

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.
- `to.EST(timestring)` converts a string (ISO format in UTC) to EST.

```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.
```

```r
#'
#' @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
#' @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 = '')
}


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

Functions to build estimate

To produce the final estimated range, we need

- `vote.diff`: calculate the difference between two snapshots for a states
- `rbind.exclude.dup`: like `rbind` but exclude row with duplicate row names. useful to combine vote differences, mail vote counts, and all vote counts to calculation probability C.I.
- `build.est`: build estimated probability given the votes count. Vote difference will be used first to show the most recent trend, then mail vote counts if vote difference is not available, and at the end all vote counts. Estimated expected vote remaining is from NYT's data.

```r
#' Calculate the vote difference between the new snapshot and the old snapshot.
#'
#' @param dataset `all.state.dataset`
#' @param state state name
#' @param index.old the index of the old snapshot
#' @param index.new the index of the new snapshot. if omitted, the latest will be used.
#' @param type either 'all.votes' or 'mail.votes'
#'
#' @return a data frame contains the vote difference for each candidate by county.
#' @export
#'
#' @examples all.state.dataset %>% vote.diff('arizona')
vote.diff <- function(dataset, state, index.old, index.new = NULL, type='all.votes') {
  if (is.null(index.new)) {
    index.new <- length(dataset[[state]])
  }
  new.votes <- get.state.data(dataset, state, index.new)[[type]]
  old.votes <- get.state.data(dataset, state, index.old)[[type]]
  return(data.frame(data.matrix(new.votes) - data.matrix(old.votes),
                    row.names = rownames(new.votes)))
}


#' Combine 2 data frame by rows,but only keep the first occurrence of 2 rows
#' with the same row name.
#'
#' @param df original data frame
#' @param df.new new data frame, only rows with new row name will be added.
#'
#' @return a new data frame
#' @export
rbind.exclude.dup <- function(df, df.new) {
  rbind(df, df.new)[!duplicated(c(rownames(df), rownames(df.new))), ]
}


#' Build estimate of a state. Candidate's probability CI will be calculated based
#' on vote changes, then mail votes, and lastly all votes.
#'
#' @param dataset all.state.dataset
#' @param state state name
#' @param index.ref index of the reference snapshot, used to get the vote difference with `index.cur`.
#'                  when omitted, the first snapshot will be used.
#' @param index.cur index of the current snapshot, used to get the current votes count and remaining vo
#'                  when omitted, the latest snapshot will be used.
#' @param alpha alpha used in `MultinomialCI::multinomialCI`
#'
#' @return a data frame contained the expected remaining votes and CI for each candidate's probability.
#' @export
build.est <- function(dataset, state, index.ref = 1, index.cur = NULL, alpha=0.05) {
  if (is.null(index.cur)) {
    index.cur <- length(dataset[[state]])
  }
```

```
  state.data.cur <- get.state.data(dataset, state, index.cur)

  using <- list(vote.diff(dataset, state, index.ref, index.cur),
                state.data.cur$mail.votes,
                state.data.cur$all.votes)

  using %>% lapply(CI.by.county) %>% reduce(rbind.exclude.dup) -> ci

  remaining <- data.frame(
    exp.remaining = pmax(0, state.data.cur$state.details$tot_exp_vote
                         - rowSums(data.matrix(state.data.cur$all.votes))),
    row.names = state.data.cur$state.details$name)

  est <- merge(remaining, ci, by=0)
  est <- est[order(est$Row.names), ]
  rownames(est) <- est$Row.names
  est$Row.names <- NULL
  return(est)
}
```

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.

```
build.final.range <- function(est) {
  future.lo <- p.interval(est$exp.remaining, est$bidenj.low)[1] - p.interval(est$exp.remaining, est$trum
  future.hi <- p.interval(est$exp.remaining, est$bidenj.high)[2] - p.interval(est$exp.remaining, est$tru
  current.diff <- sum(all.votes$bidenj) - sum(all.votes$trumpd)
  return(c(
    'est.final' = current.diff + c(future.lo, future.hi),
    'current.diff' = current.diff
  ))
}
```

Running to model

Pull data

Pull new data from a state from NYT's API.

`tracking.states` includes a vector of all states of which we want to pull data. State name should be fully spelled in lower case, and white spaces should be replaced by `-`. For example, New York will be `new-york`; Pennsylvania will be `pennsylvania`, and D.C. will be `district-of-columbia`.

```
tracking.states <- c('arizona', 'pennsylvania', 'georgia')
invisible(lapply(tracking.states,
                 function(state) (all.state.dataset %>%
                                  collect.data(state) ->>
                                  all.state.dataset)))
```

```
## arizona : No update
## Update time: 2020-11-09 13:32:33
## Current margin for Biden: 17131
## pennsylvania : No update
## Update time: 2020-11-09 13:50:38
## Current margin for Biden: 45246
## georgia : No update
## Update time: 2020-11-09 14:01:41
## Current margin for Biden: 10647
```

Set `current.state.name` for analysis.

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

```
## Get state data for arizona (updated at: 2020-11-09 13:32:33).
## State data are loaded in: state.details, all.votes, mail.votes.
```

Estimate the probability for each candidate

Note: to change the reference snapshot and the current snapshot to build the estimate, add/change the `index.ref` and `index.cur` arguments in `build.est`.

To see the number of snapshots available in each state, use `sapply(all.state.dataset, length)`.

To see the time of a state's snapshot, use `names(all.state.dataset[[current.state.name]])` (time in UTC) or `to.EST(names(all.state.dataset[[current.state.name]]))` (time in EST).

```
est <- build.est(all.state.dataset, current.state.name, )
```

```
## Get state data for arizona (updated at: 2020-11-09 13:32:33).
## Get state data for arizona (updated at: 2020-11-09 13:32:33).
## Get state data for arizona (updated at: 2020-11-06 11:09:33).
```

```
est[est$exp.remaining > 0,]
```

```
##            exp.remaining bidenj.low trumpd.low jorgensenj.low write.ins.low
## Cochise             6619  0.3209024  0.6040516    0.012891344    0.00000000
## Coconino            1587  0.4087432  0.4939891    0.000000000    0.00000000
## Gila                  47  0.2696078  0.5931373    0.000000000    0.00000000
## La Paz               823  0.3208176  0.6380804    0.000000000    0.00000000
## Maricopa           25705  0.4049520  0.5069449    0.023872897    0.05277834
## Mohave              2164  0.1632550  0.7892343    0.011119535    0.00000000
## Navajo               901  0.3095432  0.6296900    0.003262643    0.00000000
## Pima               25936  0.4263302  0.4639822    0.017780102    0.04431952
## Pinal               5476  0.3434707  0.6190833    0.017180744    0.00000000
## Santa Cruz            21  0.3636364  0.3030303    0.000000000    0.00000000
## Yavapai             2281  0.1938844  0.7382392    0.021505376    0.00000000
## Yuma                2105  0.4822620  0.4651304    0.014402668    0.00000000
##            bidenj.high trumpd.high jorgensenj.high write.ins.high
## Cochise      0.3631721   0.6463213      0.05516106    0.021551485
## Coconino     0.4765113   0.5617572      0.06339654    0.033888347
## Gila         0.4009292   0.7244586      0.07739974    0.067595822
## La Paz       0.3496789   0.6669417      0.02597304    0.015975262
## Maricopa     0.4106834   0.5126763      0.02960428    0.058509718
## Mohave       0.1879669   0.8139462      0.03583144    0.012581500
## Navajo       0.3486252   0.6687720      0.04234461    0.019913945
```

```
## Pima         0.4503695   0.4880215      0.04181940    0.068358813
## Pinal        0.3541416   0.6297541      0.02785156    0.009604223
## Santa Cruz   0.6991630   0.6385570      0.21431453    0.184011501
## Yavapai      0.2249487   0.7693035      0.05256964    0.015607279
## Yuma         0.5078621   0.4907305      0.04000282    0.012865162
```

Build the final range

```
est %>% build.final.range() -> final.range
final.range
```

```
## est.final.lo est.final.hi current.diff
##     4067.672     9730.493    17131.000
```

Saving the data

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