Index

Functions

bptool module

This module provides routines for computing the winning probabilities for various candidates, given audit sample data, using a Bayesian model, in a ballot-polling audit of a plurality election. The election may be single-jurisdiction or multi-jurisdiction. In this module we call a jurisdiction a "county" for convenience, although it may be a precinct or a state or something else, as long as you can sample from its collection of paper ballots.

The Bayesian model uses a prior pseudocount of "+1" for each candidate.

If this module is imported, rather than used stand-alone, then the procedure compute_win_probs can compute the desired probability of each candidate winning a full recount, given sample tallies for each county.

For command-line usage, there are really two modes:

(1) For single-county usage, give a command like: python bptool.py 10000 60 50 30 where 10000 is the total number of votes cast in the county 60 50 30 are the votes seen for each candidate in the auditing so far

(2) For multiple-county usage, give a command like python bptool.py --path_to_csv test.csv where test.csv is a file like: county name, total votes, Alice, Bob 1, 1000, 30, 15 2, 2000, 40, 50 with one header line, then one line per county. The field names "county name" and "total votes" are required; the candidate names are the candidate names for the contest being audited.

There are optional parameters as well, to see the documentation of them, do python bptool.py --h

More description of Bayesian auditing methods can be found in:

TOP

```
A Bayesian Method for Auditing
Elections
by Ronald L. Rivest and Emily
Shen
EVN/WOTE'12 Proceedings
http://people.csail.mit.edu
/rivest/pubs.html#RS12z

Bayesian Tabulation Audits:
Explained and Extended
by Ronald L. Rivest
2018
http://people.csail.mit.edu
/rivest/pubs.html#Riv18a

Bayesian Election Audits in One
Page
by Ronald L. Rivest
2018
http://people.csail.mit.edu
/rivest/pubs.html#Riv18b
```

SHOW SOURCE ≡

Functions

def compute_win_probs(sample_tallies,
total_num_votes, seed,
num_trials,
candidate_names,
vote_for_n)

Runs num_trials simulations of the Bayesian audit to estimate the probability that each candidate would win a full recount.

In particular, we run a single iteration of a Bayesian audit (extend each county's sample to simulate all the votes in the county and calculate the overall winner across

TOP

counties) num_trials times.

Input Parameters:

-sample_tallies is a list of lists. Each list represents the sample tally for a given county. So, sample_tallies[i] represents the tally for county i. Then, sample_tallies[i][j] represents the number of votes candidate j receives in county i.

-total_num_votes is a list of integers representing the number of ballots that were cast in this election. Each integer represents the total number of votes cast in a given county. So, total_num_votes[i] represents the total votes for county i. The sum of all total_num_votes[i] is the total number of votes in the entire election.

-seed is an integer or None. Assuming that it isn't None, we use it to seed the random state for the audit.

-num_trials is an integer which represents how many simulations of the Bayesian audit we run, to estimate the win probabilities of the candidates.

-candidate_names is an ordered list of strings, containing the name of every candidate in the contest we are auditing.

-vote_for_n is an integer, parsed from the command-line args. Its default value is 1, which means we only calculate a single winner for the election. For other values n, we simulate the unnsampled votes and define a win for candidate i as any time they are in the top n candidates in the final tally.

Returns:

-win_probs is a list of pairs (i, p) where p is the fractional representation of the number of trials that candidate i has won out of the num_trials simulations.

SHOW SOURCE ≡

TOP

```
def compute_winner(sample_tallies,
                   total_num_votes,
                   vote_for_n, seed,
                   pretty_print=False)
```

Given a list of sample tallies (one sample tally per county) a list giving the total number of votes cast in each county, and a random seed (an integer) compute the winner in a single simulation. For each county, we use the Dirichlet-Multinomial distribution to generate a nonsample tally. Then, we sum over all the counties to produce our final tally and calculate the predicted winner over all the counties in the election.

Input Parameters:

-sample_tallies is a list of lists. Each list represents the sample tally for a given county. So, sample_tallies[i] represents the tally for county i. Then, sample_tallies[i][j] represents the number of votes candidate j receives in county i.

-total_num_votes is a list of integers. Each integer represents the total number of votes cast in a given county. So, total_num_votes[i] represents the total votes for county i. The sum of all total_num_votes[i] is the total number of votes in the entire election.

-seed is an integer or None. Assuming that it isn't None, we use it to seed the random state for the audit.

-vote_for_n is an integer, parsed from the command-line args. Its default value is 1, which means we only calculate a single winner for the election. For other values n, we simulate the unnsampled votes and define a win for candidate i as any time they are in the top n candidates in the final tally.

-pretty_print is a Boolean, which defaults to False. When it's set to True, we print the winning candidate, the number of votes they have received and the final vote tally for all the candidates.

Returns:

TOP

winners is a list of integers, representing the indices of the candidate who won the election. It's size equals the vote_for_n parameter, which defaults to 1.

SHOW SOURCE ≡

def convert_int_to_32_bit_numpy_array(v)

Convert value v, which should be an arbitrarily large python integer (or convertible to one) to a numpy array of 32-bit values, since this format is needed to initialize a numpy.random.RandomState object. More precisely, the result is a numpy array of type int64, but each value is between 0 and 2**32-1, inclusive.

Example: input 2**64 + 5 yields np.array([5, 0, 1], dtype=int)

Input Parameters:

-v is an integer, representing the audit seed that's being passed in. We expect v to be non-negative.

Returns:

-numpy array created deterministically from v that will be used to initialize the Numpy RandomState.

SHOW SOURCE ≡

def create_rs(seed)

Create and return a Numpy RandomState object for a given seed. The input seed should be a python integer, arbitrarily large. The purpose of this routine is to make all the audit actions reproducible.

Input Parameters:

-seed is an integer or None. Assuming that it isn't None, we convert it into a Numpy Array.

Returns:

-a Numpy RandomState object, based on the seed, or the clock time if the seed is None.

TOP

SHOW SOURCE ≡

def dirichlet_multinomial(sample_tally, total_num_votes, rs)

Return a sample according to the Dirichlet multinomial distribution, given a sample tally, the number of votes in the election, and a random state. There is an additional pseudocount of one vote per candidate in this simulation.

Input Parameters:

-sample_tally is a list of integers, where the i'th index in sample_tally corresponds to the number of votes that candidate i received in the sample.

-total_num_votes is an integer representing the number of ballots that were cast in this election within the county.

-rs is a Numpy RandomState object that is used for any random functions in the simulation of the remaining votes. In particular, the gamma functions are made deterministic using this state.

Returns:

-multinomial_sample is a list of integers, which sums up to the total_num_votes - sample_size. The i'th index represents the simulated number of votes for candidate i in the remaining, unsampled votes.

SHOW SOURCE ≡

def generate_nonsample_tally(sample_tally, total_num_votes, seed)

Given a sample_tally, the total number of votes in an election, and a seed, generate the nonsample tally in the election using the Dirichlet multinomial distribution.

Input Parameters:

TOP

-sample_tally is a list of integers, where the i'th index in sample_tally corresponds to the number of votes that candidate i received in the sample.

-total_num_votes is an integer representing the number of ballots that were cast in this election within the county.

-seed is an integer or None. Assuming that it isn't None, we use it to seed the random state for the audit.

Returns:

-nonsample_tally is list of integers, which sums up to the total_num_votes - sample_size. The i'th index represents the simulated number of votes for candidate i in the remaining, unsampled votes.

SHOW SOURCE ≡

def main()

Parse command-line arguments, compute and print answers.

SHOW SOURCE ≡

def preprocess_csv(path_to_csv)

Preprocess a CSV file into the correct format for our sample tallies. In particular, we ignore the county name column and summarize the relevant information about the sample tallies in each county, the total number of votes in each county, and the candidate names.

Input Parameters:

-path_to_csv is a string, representing the full path to the CSV file, containing sample tallies.

Returns:

-sample_tallies is a list of lists. Each list represents the sample tally for a given county. So, sample_tallies[i] represents the tally for county i. Then, sample_tallies[i][j] represents the number of votes candidate j receives in

TOP

county i.

-total_num_votes is a list of integers representing the number of ballots that were cast in this election. Each integer represents the total number of votes cast in a given county. So, total_num_votes[i] represents the total votes for county i.

-candidate_names is an ordered list of strings, containing the name of every candidate in the contest we are auditing.

SHOW SOURCE ≡

def print_results(candidate_names, win_probs, vote_for_n)

Given list of candidate_names and win_probs pairs, print summary of the Bayesian audit simulations.

Input Parameters:

-candidate_names is an ordered list of strings, containing the name of every candidate in the contest we are auditing.

-win_probs is a list of pairs (i, p) where p is the fractional representation of the number of trials that candidate i has won out of the num_trials simulations.

-vote_for_n is an integer, parsed from the command-line args. Its default value is 1, which means we only calculate a single winner for the election. For other values n, we simulate the unnsampled votes and define a win for candidate i as any time they are in the top n candidates in the final tally.

Returns:

-None, but prints a summary of how often each candidate has won in the simulations.

SHOW SOURCE ≡

TOP