Package 'elec'

January 8, 2019

Type Package

Title Collection of Functions for Statistical Election Audits

Version 0.1.2.1

Date 2010-06-01

Author Luke Miratrix

Maintainer Luke Mirarix <luke@vzvz.org>

Description This is a bizarre collection of functions written to do various sorts of statistical election audits. There are also functions to generate simulated voting data, and simulated ``truth'' so as to do simulations to check characteristics of these methods.

License GPL (>= 2)

LazyLoad yes

Repository CRAN

Date/Publication 2019-01-08 07:55:05 UTC

NeedsCompilation no

Depends R (>= 2.10)

R topics documented:

`elec-package` *Statistical Election Audits Package*

Description

This is a collection of functions written to do various sorts of statistical election audits. There are also functions to generate simulated voting data, and simulated "truth" so as to do simulations to check charactaristics of these methods. The package includes two data sets consisting of actual reported voting results for races held November, 2008, in California. It also includes actual audit date for one of these races.

Details

Package:	elec
Type:	Package
Version:	0.1
Date:	2009-01-14
License:	GPL (>= 2)
LazyLoad:	yes

There are three general audit styles implemented in this package. For each style there are two main computational tasks provided: estimate the needed sample size and expected workload, and calculate P-values for a given audit result. The three methods are CAST (see `CAST.calc.sample` and `CAST`), the Trinomial Bound (see `tri.calc.sample` or `trinomial.bound`) , and Kaplan-Markov (KM) Bound (see `KM.calc.sample` and `KM.audit`).

The examples primarily use a data set included in the package, `santa.cruz` and `santa.cruz.audit`, which holds the ballot counts for a Santa Cruz, CA race that we audited using these methods. See `trinomial.bound` for how these data were analyzed. The `yolo` data set holds precinct level counts for a race in Yolo county.

There are also many functions allowing for construction of new audit methods and simulations. This includes methods that generate fake race data that can be used for computational simulations to assess the efficay of different auditing approaches (see, e.g., `make.sample` and `make.truth`).

The package grew out of an earlier, disorganized package that implemented general routines for election auditing. Pieces of this package are used by the aforementioned cleaner methods, but all the individual functions are still there for specific uses, such as making different tests. Start with stark.test, which has an index of these pieces in its "see also" section.

If you find yourself confused, please contact the maintainer, L. Miratrix, for help. This will help improve the clarity of the package a great deal.

Author(s)

Luke W. Miratrix

Maintainer: Luke W. Miratrix <luke@vzvz.org>

References

CAST and KM were developed by Philip B. Stark. The Trinomial bound was developed by Luke W. Miratrix and Philip B. Stark.

For general papers on election auditing see the list at http://www.stat.berkeley.edu/~stark/Vote/index.htm.

In particular, for the trinomial bound, see Luke W. Miratrix and Philip B. Stark. (2009) Election Audits using a Trinomial Bound (in press).

For the KM bound see Stark, P.B., 2009. Risk-limiting post-election audits: P-values from common probability inequalities.

For an overview of the races and the methods, see Joseph Lorenzo Hall1, Philip B. Stark, Luke W. Miratrix, Elaine Ginnold, Freddie Oakley, Tom Stanionis, and Gail Pellerin. (2009) Implementing Risk-Limiting Audits in California (in press).

Description

An `audit.plan` is returned by CAST.calc.sample, containing details of how to audit for a desired level of confidence. It has a print method for pretty output.

The `audit.plan.tri`, similarly, is an object that holds information about conduting a PPEB election audit, in particular an audit that will use the trinomial bound to analyze resultant audit data. It is what is returned by the tri.calc.sample method.

Usage

```
## S3 method for class 'audit.plan'
print(x, ...)
## S3 method for class 'audit.plan'
is(x)
## S3 method for class 'audit.plan.tri'
is(x)
## S3 method for class 'audit.plan.tri'
print(x, ...)
```

Arguments

`x`	An audit plan (or trinomial audit plan).
`...`	Unused.

Details

Theoretically, auditors will use the plan and go out and generate actual audit data. (You can fake it with simulations–see make.truth.) The audit data should be stored in a new data frame with new vote totals, or overstatements, for the candidates in the audited precincts. To convert from totals to overstatements, use `audit.totals.to.OS`. You can store that in a elec.data object under "audit", or keep it seperate.

Author(s)

Luke W. Miratrix

See Also

CAST.calc.sample tri.calc.sample

`audit.totals.to.OS` *Converting total vote counts to Over Statements*

Description

This utility function takes a collection of total votes from an audit and subtracts the originally reported totals from them to give overstatement errors (i.e., how many votes more than actual a candidate had). I.e., the overstatement error is REPORTED - ACTUAL.

Usage

```
audit.totals.to.OS(Z, audit)
```

Arguments

Z	Elec.data object holding the originally reported results
audit	A data.frame with one column per candidate that holds the totals from the audit. Each row corresponds to a precinct. Object needs a PID column with precinct ids that match the ones in Z.

Details

Make sure the audit's PID column is a character vector and not a factor. If needed, convert via `audit$PID = as.character(audit$PID)`.

Value

A new data.frame with overstatement errors.

Author(s)

Luke W. Miratrix

See Also

See AuditErrors for different ways of summarizing audit errors.

Examples

```
## Generate a fake race, a fake audit, and then compute overstatements
Z = make.sample(0.08, 150, per.winner=0.4, R=2.01)
Z
Zb = make.ok.truth(Z, num.off=150, amount.off=5)
Zb
aud = Zb$V[ sample(1:Zb$N, 10), ]
aud
audit.totals.to.OS(Z, aud )
```

AuditErrors *Functions that Compute Error Levels Given Audit Data*

Description

Calculate the error amounts for all precincts in Z that were audited from the audit data, given as overstatement errors for all candidates.

Usage

```
compute.audit.errors(Z, audit = NULL, calc.e_p = calc.pairwise.e_p, w_p =
                weight.function("no.weight"), bound.col = "tot.votes",
                err.override = NULL)
calc.pairwise.e_p(Z, audit=NULL, err.override = NULL)
calc.overstatement.e_p(Z)
```

Arguments

Z	elec.data object
err.override	Assume a baserate of this amount of error everywhere, ignoring audit data. If non-null, use this as the found error in votes rather than the actual errors found in the audit.
bound.col	This is the vector (in audit) containing the maximum number of votes possible in the various precincts.
calc.e_p	Calculate e_p or take as given.
w_p	The weight function to use to reweight the errors of precincts.
audit	The audit object, if it is not in the Z object, or if some other object other than the one in the Z object is desired to be considered as the audit object. Used by the simulation functions to generate errors for some fixed amount of error in conjunction with the err.override.

Details

compute.audit.errors uses the calc functions and the weight functions in a 1-2 combination.

calc.pairwise.e_p() is often used with an err.override for simulation studies and whatnot to see what a fixed vote impact would have on taints for trinomial.

Value

compute.audit.errors returns a new audit table from Z with two new columns, err and err.weighted, corresponding to the errors found in each audited precinct before and after the weight function has been applied to them.

Note

Z must have an audit component, or one must be passed, for this function to make sense! Remember that audit objects have overstatements, NOT total votes for candidates. With err.override being set this is less relevant as the actual votes are usually ignored.

Author(s)

Luke W. Miratrix

See Also

See audit.totals.to.OS for a utility function that handles processing of audit data.

CAST *CAST Functions*

Description

Collection of functions for planning and evaluating results of a CAST election audit. CAST is a system devised by Dr. Philip B., Stark, UC Berkeley Department of Statistics.

`CAST.calc.sample` determines what size SRS sample should be drawn to have a reasonable chance of certification if the election does not have substantial error. It returns an `audit.plan`. `CAST.sample` takes the audit.plan and draws a sample to audit. `CAST.audit` takes audit data (presumably from the audit of the sample drawn in previous step) and analyzes it.

Usage

```
CAST.calc.sample(Z, beta = 0.9, stages = 1, t = 3, as.taint=FALSE,
            small.cut = NULL, strata = NULL, drop = NULL,
            method = c("select", "binomial", "hypergeometric"),
            calc.e.max = TRUE, bound.function = maximumMarginBound)
CAST.sample(Z, ns, strata = NULL,
         seed = NULL, print.trail = FALSE,
         known = "known")
CAST.audit(Z, audit = NULL, plan = NULL, ...)
test.CAST()
```

Arguments

`Z`	elec.data object
`beta`	the confidence level desired
`stages`	number of auditing stages. Each stage will have the same confidence level, determined by a function of beta. A value of 1 is a single-stage audit.
`t`	The maximum amount of error, in votes, expected.
`as.taint`	Boolean value. TRUE means interpret t as a taint in $[0,1]$ by batch (so the threshold error will be batch-specific). FALSE means interpret t as a proportion of the margin or as number of votes (as described above).
`small.cut`	Cut-off in votes–any precincts with potential error smaller than this value will not be audited and be assumed to be worst case error.
`strata`	Name of the stratification column of Z. Not needed if audit plan also being passed in case of CAST.sample. NULL means single strata.
`drop`	Vector of precincts to drop for whatever reasons (such as they are already known).
`method`	Method of calculation.
`calc.e.max`	Should the e.max be taken as given, or recalculated?
`bound.function`	What function should be used to calculate worst-case potential error of precincts.

ns	EITHER an audit.plan or a vector of sample sizes for the strata. Names must correspond ot the names of the strata. If ns is an audit plan, then the strata variable should not be passed as well.
seed	Seed to use–for reproducability.
print.trail	Print out diagnostics.
known	The column of known precincts that should thus not be selected. Similar to "drop", above.
plan	An audit.plan object that the audit was conducted under.
audit	A data.matrix holding the audit data, if the Z object does not have one, or if it is desirable to override it. If both the Z object has an audit object and audit is not null, it will use this parameter and ignore the one in Z.
...	Passed to CAST.calc.sample if plan is null and needs to be regenerated.

Author(s)

Luke W. Miratrix

References

Philip B. Stark. CAST: Canvass Audits by Sampling and Testing. University of California at Berkeley Department of Statistics, 2009. URL: http://statistics.berkeley.edu/~stark/Preprints/cast09.pdf. Also see http://www.stat.berkeley.edu/~stark/Vote/index.htm for other relevant information.

See Also

elec.data for a description of the object that holds precinct-level vote records. See tri.calc.sample for a PPEB auditing method. See CAST.calc.opt.cut for calculating optimal cut-offs to keep needed sample size low. Also see sim.race, do.audit, make.sample, and make.truth for doing simulation studies of this method.

Examples

```
        ## Make an example cartoon race (from Stark paper)
Z = make.cartoon()

        ## What should we do?
samp.info = CAST.calc.sample( Z )
samp.info

        ## Draw a sample.
samp = CAST.sample( Z, samp.info$ns )
        samp

        ## Analyze what a CAST audit of santa cruz would entail
        data(santa.cruz)
        Z = elec.data( santa.cruz, C.names=c("leopold","danner") )
        CAST.calc.sample( Z, beta=0.75, stages=1, t=5, small.cut=60)
```

`CAST.calc.opt.cut` *Calculate Optimal CAST plan*

Description

With CAST, it is sometimes advantageous to set aside small precincts and assume they are entirely in error so as to reduce the total number of precincts in the pool that we sample from. This trade-off can increase the power of the audit or, in other terms, allow us to sample fewer precincts as the chance of nabbing the large, dangerous ones is larger.

Of all cuts that produce the smallest `n`, it returns the smallest cut (since sometimes multiple cut-offs lead to the same sample size).

This function also plots the trade-off of sample size for a specific cut, if the plot flag is TRUE.

Usage

```
CAST.calc.opt.cut(Z, beta = 0.9, stages = 2, t = 3, plot = FALSE, ...)
```

Arguments

`Z`	The elec.data object
`beta`	1-`beta` is the risk of the audit failing to notice the need to go to a full manual count if it should.
`stages`	Number of stages in the audit.
`t`	The allowed vote swing that is not considered a material error.
`plot`	TRUE/FALSE. Plot the trade-off curve.
`...`	Extra arguments to the plot command.

Details

This function iteratively passes increasing values of `small.cut` to `CAST.calc.sample` and examines the resulting `n`.

Value

Returns a list.

`cut`	Size of the optimal cut. All precincts with an error smaller than or equal to cut would not be audited, and instead be assumed to be in full error.
`n`	Corresponding needed sample size given that cut.
`q`	The number of tainted precincts that would be needed to throw the election, beyond the ones set aside due to being smaller than `cut`.

Author(s)

Luke W. Miratrix

Examples

```
## Find optimial cut for  determining which small precincts that
## we would set aside and not audit in Santa Cruz
data(santa.cruz)
Z = elec.data( santa.cruz, C.names=c("leopold","danner") )

CAST.calc.opt.cut( Z, beta=0.75, stages=1, t=5, plot=TRUE )
```

`compute.stark.t` *compute.stark.t*

Description

Compute the test statistic for election audits, essentially the largest error found in the audit, as measured by the passed functions and methods.

This is an older method that other methods sometime use—it is probably best ignored unless you have a good reason not to.

Usage

```
compute.stark.t(Z, bound.col, calc.e_p = calc.pairwise.e_p,
           w_p = weight.function("no.weight"),
           err.override = NULL,
           return.revised.audit = FALSE)
```

Arguments

`Z`	If it already has an audit table with err and err.weighted then it will use those errors, otherwise it will compute them with compute.stark.err
`bound.col`	This is the vector containing the maximum number of votes possible in the various precincts.
`calc.e_p`	Function to compute e_p. Default is calc.pairwise.e_p.
`w_p`	The weight function to be applied to the precinct error.
`err.override`	If non-null, use this as the found error in votes rather than the actual errors found in the audit.
`return.revised.audit`	Return the updated audit frame with the error and weighted errors calculated.

Value

The test statistic, i.e. the maximum found error in the audit sample, as computed by calc.e_p and weighted by w_p.

Author(s)

Luke W. Miratrix

See Also

find.q stark.test

---

countVotes *countVotes*

Description

Given a elec.data object, count the votes as reported and determine winner(s) and loser(s).

Usage

```
countVotes(Z)
```

Arguments

Z	the elec.data object.

Value

Updated 'Z' matrix with the total votes as components inside it.

Author(s)

Luke W. Miratrix

Examples

```
Z = make.cartoon()
## Take away 20 percent of C1's votes.
Z$V$C1 = Z$V$C1 * 0.8
## Count again to find winner.
Z = countVotes(Z)
Z
```

---

do.audit *do.audit*

Description

Given a list of precincts to audit, the truth (as an elec.data object), and the original votes (also as an elec.data object), do a simulated CAST audit and return the audit frame as a result.

Usage

```
do.audit(Z, truth, audit.names, ns = NULL)
```

Arguments

Z	elec.data object
truth	another elec.data object–this one's vote counts are considered "true"
audit.names	name of precincts to audit. Correspond to rownames of the Z and truth elec.data objects.
ns	List of sample sizes for strata. If this is passed, this method will randomly select the precincts to audit. In this case audit.names should be set to NULL.

Details

Given the reported vote table, Z, and the actual truth (simulated) (a Z matrix with same precincts), and a list of precincts to audit, do the audit. If audit.names is null and the ns is not null, it will sample from precincts via CAST.sample automatically.

Value

Overstatments for each candidate for each precinct.

Author(s)

Luke W. Miratrix

See Also

CAST for how to run the CAST auditing method. See `make.sample` and `make.truth` for generating fake situations for doing simulation studies of the CAST method. See AuditErrors and `audit.totals.to.OS` for utility functions handing processing of audit data.

Examples

```
Z = make.cartoon(n=200)
truth = make.truth.opt.bad(Z, t=0, bound="WPM")
samp.info=CAST.calc.sample(Z, beta=0.75, stages=1, t=5 )
audit.names = CAST.sample( Z, samp.info )
do.audit( Z, truth, audit.names )
```

`elec.data` *core election audit data structure*

Description

Makes an object (often called a 'Z' object in this documentation) that holds all the vote totals, etc., as well as some precomputed information such as vote margins between candidates, the theoretical winners, and so on.

`elec.data` does some cleaning and renaming of the passed data structure. In particular it will rename the tot.votes column to "tot.votes" if it is not that name already.

`make.Z` just passes all arguments to `elec.data()`—it is the same thing. It is the original name of elec.data and is included for legacy and nostalgia reasons.

Usage

```
elec.data(V, C.names = names(V)[2:length(V)],
          f = 1, audit = NULL,
          pool = TRUE, tot.votes.col = "tot.votes", PID.col="PID")
make.Z( ... )
## S3 method for class 'elec.data'
is(x)
## S3 method for class 'elec.data'
print(x, n = 4, ...)
```

Arguments

`V`	Voter matrix OR 2-element list with Voter Matrix followed by Candidate names
`C.names`	List of candidate names. Also names of columns in V
`f`	Number of winners
`audit`	The audit data—must have columns that match C.names. Columns are overstatements of votes found for those candidates.
`pool`	Combine small candidates into single pseudo-candidates to increase power
`tot.votes.col`	Name of column that has the total votes for the precincts.
`PID.col`	Name of column that identifies unique PIDs for precincts.
`...`	The collection of arguments that are passed directly to elec.data, or (in the case of print), unused.
`x`	For print() and is.elec.data(). An elec.data object
`n`	For print(). number of sample precincts to print

Value

A "elec.data" data structure. Note: Will add PID (precinct ID) column if no PID provided (and generate unique PIDs). It will rename the PID column to PID. Also, rownames are always PIDs (so indexing by PID works).

Author(s)

Luke W. Miratrix

See Also

See CAST for the CAST method. See `tri.calc.sample`, `tri.sample`, and `audit.plan.tri` for the trinomial bound method. See `countVotes` for counting the votes listed in Z.

Examples

```
data(santa.cruz)
elec.data( santa.cruz, C.names=c("danner","leopold") )
```

`find.q` *find.q*

Description

Find q, the minimum number of precints with w_p's greater than given t.stat that can hold an entire election shift in them.

This number is behind the SRS methods such as CAST. If we know how many precincts, at minimum, would have to hold substantial error in order to have the reported outcome be wrong, we can compute the chance of finding at least one such precinct given a SRS draw of size n.

Usage

```
find.q(V, t.stat, bound.col, M, threshold = 1,
          w_p = weight.function("no.weight"), drop = NULL)
```

Arguments

`V`	The data.frame of votes–the subwing of a elec.data object, usually.
`t.stat`	The worst error found in the audit (weighted, etc.)
`bound.col`	The name of the column in V to be used for the passed size (max number of votes, total votes, incl undervotes, etc.) to the error function.
`M`	The margin to close. Usually 1 for proportional. Can be less if error from other sources is assumed.
`threshold`	The total amount of error to pack in the set of tainted precincts
`w_p`	The weight function for errors.
`drop`	Drop precincts with this column having a "true" value–they are previously audited or otherwise known, and thus can't hold error. Can also pass a logical T/F vector of the length of nrow(V)

Details

Find the number of precints that need to have "large taint" in order to flip the election. This is, essentially, finding a collection of precints such that the max error (e.max) plus the background error (the w_p-inverse of the t.stat) for the rest of the precints is greater than the margin (or 1 if done by proportions).

Value

integer, number of badly tainted precints needed to hold 'threshold' error

Author(s)

Luke W. Miratrix

`find.stark.SRS.p` *find.stark.SRS.p*

Description

Find the p-value for a given q, n, and N. Helper function for a simple hypergeometric calculaton–see reports.

Usage

```
find.stark.SRS.p(N, n, q)
```

Arguments

`N`	total number of precints
`n`	total number of audited precints (must be less than N)
`q`	min number of precints that could hold taint to flip election

Value

Chance that 1 or more of the q 'bad' things will be seen in a size n SRS draw from the N sized bucket.

Author(s)

Luke W. Miratrix

find.stratification *find.stratification*

Description

Find how audit covered the strata for a given table of votes and audits.

Usage

```
find.stratification(D, aud, strat.col)
```

Arguments

D	Table of votes
aud	Table of audit data
strat.col	The column to use that identifies the stratification levels

Value

Table of strata. For each stratum (row) the table has the name of the stratam, the number of precincts in the stratum, the number of audited precincts and percent of precincts audited.

Author(s)

Luke W. Miratrix

KM.audit *KM Audit Calculator*

Description

Do a KM audit given a specified list of audited batches for a specified election.

Usage

```
KM.audit(data, U, Z, alpha = 0.25, plot = FALSE, debug = FALSE, return.Ps = FALSE, truncate.Ps = TRUE)
```

Arguments

data	Data frame holding audit data with taint and tot.votes as two columns.
U	Maximum total error bound (sum of e.max for all batches in race).
Z	elec.data object for the race—the original reported results.
alpha	Risk.
plot	Plot the audit?
debug	Print debugging info
return.Ps	Return the stepwise P-values
truncate.Ps	Return the stepwise P-values only up to the audit stop point.

Details

This will do a single-stage KM audit as a consequence of doing the stepwise version (since the single-stage is the same as the stepwise up to the number of batches audited).

WARNING: This function is not fully debugged!

Value

List of various things, including final p-value.

Author(s)

Miratrix

References

Stark, Miratrix

KM.calc.sample *Calculate sample size for KM-audit.*

Description

Calculate the size of a sample needed to certify a correct election if a KM audit is planned.

Usage

```
KM.calc.sample(Z, beta = 0.75, taint = 0, bound = c("e.plus", "WPM", "passed"))
## S3 method for class 'audit.plan.KM'
print(x, ...)
```

Arguments

Z	elec.data object
beta	Desired level of confidence. This is 1-risk, where risk is the maximum chance of not going to a full recount if the results are wrong. Note that in Stark's papers, the value of interest is typically risk, denoted $alpha$.
taint	Assumed taint. Taint is assumed to be the taint for all batches (very conservative). If taint=0 then we produce a good baseline.
bound	Type of bound on the maximum error one could find in a batch.
x	A audit.plan.KM object, such as one returned by KM.calc.sample.
...	Unused.

Value

A audit.plan.KM object.

Author(s)

Based on the KM audit by Stark.

See Also

KM.audit

Examples

```
data(santa.cruz)
Z = elec.data( santa.cruz, C.names=c("danner","leopold") )
KM.calc.sample( Z, beta=0.75, taint=0 )
```

---

make.audit *make.audit functions*

Description

Functions that make fake audits given a specified error mechanism and a elec.data object holding reported outcomes.

Usage

```
make.audit(Z = NULL, method = c("tweak", "opt.bad", "opt.bad.WPM", "opt.bad.packed", "opt.bad.packed.
make.audit.from.Z(Z, N = 400, ...)
```

Arguments

Z	elec.data object. For make.audit.from.Z, this is the large election, holding precincts with size, votes, etc., that get sampled to make an election of a requested number of batches.
method	the method of error generation. if "tweak" (the default), then add random amounts of swing to some precincts, and call that the "truth". The other methods generate the truth according to various metrics.
p_d	percent chance of error in precinct (for ok method)
swing	vote swing if batch has error (for ok method)
max.taint	maximum taint allowed in batch
print.race	print info on race to command line?
N	The desired size of the new election.
...	other arguments to the method functions

Details

make.audit is to make the election results that can be sampled from with the simulator. This method generates the true taint and sampling weights of all precincts in the race. The taint is in column 'taint', sampling weights in 'e.max'

make.audit.from.Z Given the structure of some large election, make a small election by sampling batches (with replacement) from the full list. This first samples N precincts (and gets the totals from them) and then builds the 'truth' as normal using the make.audit() method. Note different calls to this will produce different margins based on precincts selected.

WARNING: It is concievable that the winner will flip due to the sampling, if the sample has too many batches for the loser.

Value

Data frame with precinct information for the race. NOTE- The reported vote totals are just that, reported.

Author(s)

Miratrix

See Also

truth.looker

`make.opt.packed.bad` *make.truth.opt.bad*

Description

Generate a "truth" that is optimally bad in the sense of the margin in error is packed into as few precints as possible.

Usage

```
make.opt.packed.bad(Z, max.taint = 1, max.taint.good = max.taint, WPM = FALSE, add.good = 0, add.randc
```

Arguments

`Z`	elec.data object to make bad truth for.
`max.taint`	max taint for any batch
`max.taint.good`	max taint in good direction for any batch
`WPM`	Use WPM bound on error.
`add.good`	add this amount of margin in good error (i.e. for the winner)
`add.random`	add a random tweak to error

Details

Make an audit data.frame with the error being exactly 1 margin, and packed into a small number of precincts (with some potential for binding amount of error per precinct).

Warning: error is not necessarily achievable as the discrete nature of whole votes is disregarded.

Value

Return the vote matrix (a data.frame) with tot.votes, e.max, and taint computed (NOT the elec data object).

`make.random.truth` *making fake truth for electios*

Description

Make a random truth that is with the reported outcome, but has random error scattered throughout.

Usage

```
make.random.truth(Z, p_d = 0.1, swing = 10, uniform = TRUE, seed = NULL, PID = "PID")
```

Arguments

`Z`	elec.data object. The original reported results.
`p_d`	chance a batch has error
`swing`	max amount of error in votes.
`uniform`	if yes, then error is from 1 to swing. If no, then error is swing.
`seed`	random seed to ease replication
`PID`	which column has batch IDs.

Details

Given reported results (Z), make a new data.frame which is the truth (that can be 'audited' by looking at relevant precincts).

This is the generic small error generation used in trinomial paper and elsewhere as a baseline "normal" mode of operations.

Value

Return: elec.data object holding the 'truth'.

make.sample *make.sample and friends*

Description

These methods are for SIMULATION STUDIES. These functions will build a sample, i.e. simulated, record of votes given certain parameters.

Usage

```
make.cartoon(n = 400, vote.dist = c(125, 113, 13),
             stratify = TRUE)
make.sample(M, N, strata = 1, per.winner = NULL,
                 worst.e.max = NULL, R = NULL,
                 tot.votes = 1e+05)
make.sample.from.totals(vote.W, vote.L, totals)
make.sample.from.totals.margin(M, totals, per.winner = NULL)
```

Arguments

M	The margin desired between the winner and loser (as a percent).
N	Number of precincts desired.
strata	Number of strata desired.
per.winner	The percent of votes the winner should receive.
worst.e.max	The worst e.max possible for any precinct.
R	The "dispersion" a measure of how unequal in size precincts should be. R needs to be greater than 0. NULL indicates equal size. For R between 0 and 1, the precincts are distributed 'linearly', i.e., the size of precinct i is proportional to i. At 2, the smallest precint will be near 0 and the largest twice the average votes per precinct. After 2, the precincts are distributed in a more curved fashion so that the smaller precincts do not go negative.
tot.votes	The total votes desired.
vote.W	Total votes for winner.
vote.L	Total votes for loser.
totals	Vector of total votes for precincts.
vote.dist	reported votes for C1, C2, and C3 in order for all precincts.prompt
n	Size of sample.
stratify	Should the sample be stratified?

Details

make.cartoon() makes the sample scenario described in Stark's CAST paper.

Value

A elec.data object meeting the desired specifications.

Author(s)

Luke W. Miratrix

References

See http://www.stat.berkeley.edu/~stark/Vote/index.htm for relevant information.

See Also

`elec.data` `make.truth` `do.audit`

Examples

```
Z = make.sample(0.08, 150, per.winner=0.4)
Z

Z2 = make.sample(0.08, 150, per.winner=0.4, R=2.2)
Z2

## Note how they have different precinct sizes.

summary(Z$V$tot.votes)
summary(Z2$V$tot.votes)
```

`make.truth.x` *make.truth.X*

Description

For simulations. These methods, given an elec.data object, make a "truth"—i.e. a different vote count—that meets the same precinct and tot.votes structure, but has potentially different results and outcomes.

`make.truth.opt.bad` makes the "optimally worse truth", where the error needed to flip the winner and runner-up is packed into as a few precincts as possible.

`make.ok.truth` makes the truth have the same outcome as the reported, but some errors here and there.

Usage

```
make.truth.opt.bad(Z, strata = "strata",
                bound = c("margin", "WPM"), t = 0)
make.truth.opt.bad.strat(Z, strata = "strata",
        t = 3, shuffle.strata = FALSE)
make.truth.ex.bad(Z)
make.ok.truth(Z, num.off = 8, amount.off = 5)
```

Arguments

Z	The elec.data to build from.
strata	name of column holding strata, if any.
bound	What sort of maximum error can be held in a precinct.
t	Number of per-precinct vote "background" error that can occur without triggering escalation if seen.
shuffle.strata	Should the error be randomly put in the strata?
num.off	Number of precincts that should have small errors. Direction of errors split 50-50 positive and negative.
amount.off	Size of the small errors that should be imposed.

Value

Another elec.data matrix with the same candidates and total ballot counts as the passed frame, but with different candidate totals and by-precinct votes. Can be used to test the power or actual confidence of the various auditing procedures.

WARNING: make.ok.truth randomly adds votes and can thus sometimes exceed the allowed ballot count for a precinct by small amounts.

WARNING: If the desired bound is WPM, the error in make.opt.bad.truth is made by simply adding the maximum allowed amount of error in votes to the first loser's total (so that total votes may in this case exceed the total votes of the precinct)–this could potentially cause trouble. Be careful!

WARNING: `make.truth.ex.bad` and `make.truth.opt.bad.strat` only work in conjunction with the `make.cartoon` method.

Author(s)

Luke W. Miratrix

See Also

`elec.data` `make.sample` `do.audit` `make.cartoon`

Examples

```
## First make a fake election.
Z = make.sample(0.08, 150, per.winner=0.4, R=2.2)
Z
```

```
## Now make a fake truth, which has a lot of small errors:
Zb = make.ok.truth(Z, num.off=150, amount.off=5)
Zb

## Finally, make the hardest to detect (via SRS) ``wrong'' election:
Zw = make.truth.opt.bad( Z, t=4 )
Zw
```

`marin` *Marin Measure B Reported Results*

Description

These are the reported vote totals from the 2009 election in Marin, CA for Measure B.

Usage

```
data(marin)
```

Format

A data frame with 544 observations on the following 5 variables.

`PID` Batch ID

`strata` There are two levels, `ST-IB` `ST-VBM` for in-precinct and Vote-by-Mail.

`tot.votes` total ballots cast in the batch.

`Yes` Number recorded for Yes

`No` Number recorded for No

Details

Note the vote totals for the VBM strata are made up. The batches are the "Decks", which could not be individually tallied with ease. The work-around was complex. See the references, below.

Source

Marin, CA 2009 reported election results.

References

See J. L. Hall, L. W. Miratrix, P. B. Stark, M. Briones, E. Ginnold, F. Oakley, M. Peaden, G. Pellerin, T. Stanionis, and T. Webber. Implementing risk-limiting audits in california. USENIX EVT/WOTE in press, July 2009.

Examples

```
data(marin)
marin = elec.data( marin, C.names=c("Yes","No") )

# Hand fixing error bound due to unknown
# vote totals in the VBM decks
marin$V$e.max = maximumMarginBound(marin)
sum( marin$V$e.max )   # 7.128
vbm = marin$V$strata=="ST-VBM"
marin$V[ vbm, "e.max" ] = 2 * marin$V[ vbm, "tot.votes" ] / marin$margin

sum( marin$V$e.max )   # 9.782
```

`maximumBounds` *Election Audit Error Bound Functions*

Description

Various bounding functions used to bound the maximum amount of error one could see in a single audit unit.

Usage

```
maximumMarginBound(Z, votes = NULL)
fractionOfVotesBound(Z, frac = 0.4)
```

Arguments

`Z`	The elec.data object.
`votes`	The data.frame to compute the maximumMarginBounds for. If null, will return all bounds for all precincts in Z.
`frac`	Fraction of total votes that could be a winner overstatement/loser understatement. So if the worst-case is a 20% flip then enter 0.4

Details

maximumMarginBound return the maximum margin reduction for each precint by computing all margin reductions between pairs of winners & losers and then scaling by that pair's total margin to get a proportion and then taking the max of all such proportions (usually will be the last winner to the closest loser).

fractionOfVotesBound: WPM. The maximum error of the unit is a fixed percentage of the total votes cast in the unit. Typically the 20% WPM is used–meaning a swing of 40% is the largest error possible as 20% of the votes go from the winner to the loser.

Value

Vector (of length of precincts) of maximum possible error for each precinct.

Author(s)

Luke W. Miratrix

`opt.sample.size` *KM Audit Sample Size Calc*

Description

Calc KM Optimal Sample Size

Usage

```
opt.sample.size(Z, beta = 0.25)
```

Arguments

`Z`	elec.data object
`beta`	risk

Details

This is how many steps would be needed if no error was found with each step. Obviously a bit idealistic, but still useful.

Value

Single number of batches to sample.

`santa.cruz` *Santa Cruz Election Data*

Description

`santa.cruz` and `santa.cruz.audit` hold data from a Santa Cruz County, CA, contest held in November, 2008, for County Supervisor in the 1st District. The competitive candidates were John Leopold and Betty Danner. According to the semi-official results provided to us by the Santa Cruz County Clerk's office, Leopold won with votes on 45% of the 26,655 ballots. Danner received the votes on 37% of the ballots. The remaining ballots were undervoted, overvoted, or had votes for minor candidates.

Usage

```
data(santa.cruz)
```

Format

A data frame with 152 observations on the following 5 variables.

`PID` Precinct IDs (unique) for all precincts involved in race

`r` Total number of registered voters in the precinct.

`tot.votes` Total number of ballots cast in the precinct.

`leopold` Total number of ballots marked for John Leopold.

`danner` Total number of ballots marked for Betty Danner.

Details

`santa.cruz` holds the semi-official results for the race. `santa.cruz.audit` holds the audit totals for the random sample of precincts selected for the audit. Note the `santa.cruz.audit` vote counts are larger for some precincts due the missing provisional ballot counts in the semi-official results.

Source

Santa Cruz County, CA, Clerk Gail Pellerin, and their staff.

See Also

santa.cruz.audit

Examples

```
data(santa.cruz)
elec.data( santa.cruz, C.names=c("danner","leopold") )
```

`santa.cruz.audit` *Santa Cruz Election Data*

Description

`santa.cruz` and `santa.cruz.audit` hold data from a Santa Cruz County, CA, contest held in November, 2008, for County Supervisor in the 1st District. The competitive candidates were John Leopold and Betty Danner. According to the semi-official results provided to us by the Santa Cruz County Clerk's office, Leopold won with votes on 45% of the 26,655 ballots. Danner received the votes on 37% of the ballots. The remaining ballots were undervoted, overvoted, or had votes for minor candidates.

Usage

```
data(santa.cruz.audit)
```

Format

A data frame with 16 observations on the following 4 variables.

`PID` Precinct IDs (unique) for all precincts involved in race

`leopold` Total number of ballots marked for John Leopold.

`danner` Total number of ballots marked for Betty Danner.

`count` The number of times precinct was sampled in the PPEB sample taken.

Details

`santa.cruz.audit` holds the audit totals for the random sample of precincts selected for the audit. Note the `santa.cruz.audit` vote counts are larger for some precincts due the missing provisional ballot counts in the semi-official results.

Source

Santa Cruz County, CA, Clerk Gail Pellerin, and their staffs, which we thank for their generous cooperation and the onsiderable time and effort they spent counting ballots by hand in order to collect these data.

See Also

santa.cruz. For an illustration of analyzing this data, see the example in `trinomial.bound`.

Examples

```
data(santa.cruz.audit)
data(santa.cruz)
santa.cruz = elec.data(santa.cruz, C.names=c("leopold","danner"))
trinomial.audit( santa.cruz, santa.cruz.audit )
```

---

sim.race *Simulate CAST audits to assess performance*

Description

Simulate a race (using the `make.cartoon` method) and run a CAST audit on that simulation. CAST is a system devised by Dr. Philip B., Stark, UC Berkeley Department of Statistics.

Usage

```
sim.race(n = 800, beta = 0.75, stages = 2,
         truth.maker = make.truth.opt.bad,
         print.trail = FALSE)
```

Arguments

beta	the confidence level desired
stages	number of auditing stages. Each stage will have the same confidence level, determined by a function of beta.
print.trail	Print out diagnostics.
n	Desired sample size.
truth.maker	Function to generate "truth"

Value

A vector of 3 numbers. The first is the stage reached. The second is the total number of precincts audited. The third is 0 if the audit failed to certify (i.e. found large error in the final stage), and 1 if the audit certified the election (did not find large error in the final stage).

Author(s)

Luke W. Miratrix

References

See http://www.stat.berkeley.edu/~stark/Vote/index.htm for relevant information.

See Also

See CAST and CAST.calc.opt.cut for methods regarding CAST audits. Also see do.audit, make.sample, and make.truth for doing other simulation studies of this method.

Examples

```
## See how many times the CAST method fails to catch a wrong
##  election in 20 trials.
replicate( 20, sim.race( beta=0.75, stages=2, truth.maker=make.truth.opt.bad) )

## Now see how much work the CAST method does for typical elections.
replicate( 20, sim.race( beta=0.75, stages=2, truth.maker=make.ok.truth) )
```

---

simulateIt *simulate KM audits*

Description

This takes an election and a truth and conducts a KM audit.

Usage

```
simulateIt(data, M = 50, alpha = 0.25, plot = FALSE, debug = FALSE, return.Ps = FALSE, truncate.Ps = TR
```

Arguments

data	a data frame, one row per patch, with: tot.votes, e.max, taint
M	the maximum number of samples to draw before automatically escalating to a full recount.
alpha	level of risk.
plot	plot a chart?
debug	debug diag printed?
return.Ps	Return the sequence of p-values all the way up to N.
truncate.Ps	Return Ps only up to where audit stopped.

Details

Given a list of all precincts and their true taints and their sampling weights (in data, a data.frame), do a sequential audit at the specified alpha.

Value

stopPt - number of draws drawn n - number of unique precincts audited

---

stark.test *stark.test*

Description

These main methods conduct the test of the election audit and returns a p-value and other related info on that test.

It is an older method. Most likely CAST.audit or trinomial.audit should be used instead.

Usage

```
stark.test(votes, audits, C.names = NULL,
                f = 1, pool = TRUE, pairwise = FALSE, ...)
stark.test.Z(Z, calc.e_p = calc.pairwise.e_p,
        w_p = weight.function("no.weight"),
        max_err = maximumMarginBound, bound.col = Z$tot.votes.col,
        strat.col = NULL, drop = NULL,
        strat.method = NULL, err.override = NULL,
        n = NULL, t = NULL, q = NULL)
stark.pairwise.test(votes, audits, C.names = NULL,
        f = 1, pool = TRUE, ...)
```

Arguments

votes	data.frame of votes. Each row is precinct.
audits	data.frame of audits. Each row is precinct. Table reports overstatement by candidate.
C.names	Names of candidates (and names of cor columns in votes and audits tables. If NULL will derive from cols 2 on of votes
f	The number of winners
pool	If TRUE, combine small candidates into single pseudo-candidates to increase power
pairwise	if TRUE then do a pairwise test for all pairs and return highest p-value
Z	The object holding all the voting information. See below for details.
calc.e_p	The Function used to calculate maximum error bounds
w_p	The function used to calculate weights of error (A list of two functions)
max_err	Function to compute max error bounds for each precint
bound.col	Name (or column index) of column in the vote matrix corresponding to maximum number of votes allowed in precinct.
strat.col	Name of column that determines how to stratify if NULL will not stratify
strat.method	Not currently implemented.
err.override	If non-null, use this as the found error in votes rather than the actual errors found in the audit.
n	Elements of the test statistic. Can pass to avoid computation if those values are already known (e.g., for a simulation)
t	Elements of the test statistic. Can pass to avoid computation if those values are already known (e.g., for a simulation)
q	Elements of the test statistic. Can pass to avoid computation if those values are already known (e.g., for a simulation)
drop	Either a vector of TRUE/FALSE or a name of a column in Z\$V of T/F values. Precincts identified by drop will be dropped from calculations.
...	Extra arguments passed directly to the work-horse method stark.test.Z

Details

stark.test() will do the entire test. It is basically a driver function that sets up 'Z' matrix and passes buck to the stark.test.Z

The Z object, in particular has: Z\$V: The table of reported votes Z\$audit: The table of audits as differences from recorded votes

Value

Return an htest object with pvalue, some relevant statistics, and the Z object used (possibly constructed) that produced those results.

Author(s)

Luke W. Miratrix

See Also

See elec.data for description of the main object. See find.q and compute.stark.t for the main components of this test. find.stark.SRS.p is a utility function for computing a p-value for a specific situation. See weight.function for functions used to weight audit errors. See MaximumBounds for different bounds on error that one might use for these tests. See find.stratification for a utility for stratification.

Examples

```
## pretending that santa cruz audit was a SRS audit (which it was not)
data(santa.cruz)
Z = elec.data(santa.cruz, C.names=c("leopold","danner"))
data(santa.cruz.audit)
## do some work to get the audit totals to overstatements
rownames(santa.cruz.audit) = santa.cruz.audit$PID
Z$audit = audit.totals.to.OS(Z, santa.cruz.audit)
Z$audit
stark.test.Z(Z)
```

---

tri.audit.sim *tri.audit.sim*

Description

This is a SIMULATION FUNCTION, and is not used for actual auditing of elections.

Given a matrix of votes, calculate the weights for all precincts and then draw a sample (using tri.sample). Then, assuming that p_d percent of the precincts (at random) have error, and the errors are due to vote miscounts of size 'swing', conduct a simulated "audit", returning the found descrepancies.

Usage

```
tri.audit.sim(Z, n, p_d = 0.1, swing = 5,
              return.type = c("statistics", "taints", "precinct"),
              seed = NULL, PID = "PID", ...)
```

Arguments

Z	elec.data object.
n	Sample size to draw.
p_d	The probability of a precinct having an error.
swing	The size of the error, in votes.

return.type	What kind of results to return: "statistics","taints", or "precinct"
seed	Random seed to use.
PID	Column name of column holding unique precinct IDs
...	Extra arguments passed to tri.sample

Value

List of taints found in such a circumstance OR precincts selected with relevant attributes (including simulated errors, if asked) OR the number of non-zero taints and the size of largest taint.

Author(s)

Luke W. Miratrix

See Also

elec.data for the object that holds vote data. See tri.calc.sample for computing sample sizes for trinomial bound audits.

Examples

```
data(santa.cruz)
Z = elec.data(santa.cruz, C.names=c("leopold","danner"))
Z$V$e.max = maximumMarginBound( Z )
## Sample from fake truth, see how many errors we get.
tri.audit.sim( Z, 10,  p_d=0.25, swing=10, return.type="precinct" )

## what does distribution look like?
res = replicate( 200, tri.audit.sim( Z, 10,  p_d=0.25, swing=10 ) )
apply(res,1, summary)
hist( res[2,], main="Distribution of maximum size taint" )
```

---

tri.calc.sample *Calculate needed sample size for election auditing using the Trinomial Bound*

Description

Calculate an estimated sample size to do a trinomial bound that would have a specified power (the chance to certify assuming a given estimate of low-error error rate), and a specified maximum risk of erroneously certifying if the actual election outcome is wrong.

Usage

```
tri.calc.sample(Z, beta = 0.75, guess.N = 20,
             p_d = 0.1, swing = 5, power = 0.9,
             bound = c("e.plus", "WPM", "passed"))
```

Arguments

Z	elec.data object
beta	1-beta is the acceptable risk of failing to notice that a full manual count is needed given an election with an actual outcome different from the semi-official outcome.
guess.N	The guessed needed sample size.
p_d	For the alternate: estimate of the proportion of precincts that have error.
swing	For the alternate: estimate of the max size of an error in votes, given that error exists.
power	The desired power of the test against the specified alternate defined by p_d and swing.
bound	e.plus, WPM, or use the passed, previously computed, e.max values in the Z object.

Value

An audit.plan.tri object. This is an object that holds information on how many samples are needed in the audit, the maximum amount of potential overstatement in the election, and a few other things.

Author(s)

Luke W. Miratrix

References

See Luke W. Miratrix and Philip B. Stark. (2009) Election Audits using a Trinomial Bound. http://www.stat.berkeley.edu/~stark

See Also

See elec.data for information on the object that holds vote counts. See tri.sample for drawing the actual sample. See audit.plan.tri for theo object that holds the audit plan information (e.g., number of draws, estimated work in ballots to audit, etc.). See trinomial.bound for analyzing the data once the audit results are in. See tri.audit.sim for simulating audits using this method. See CAST for an SRS audit method.

Examples

```
data(santa.cruz)
Z = elec.data( santa.cruz, C.names=c("danner","leopold") )
tri.calc.sample( Z, beta=0.75, guess.N = 10, p_d = 0.05,
               swing=10, power=0.9, bound="e.plus" )
```

`tri.sample` *Sample from List of Precincts PPEB*

Description

tri.sample selects a sample of precincts PPEB. Namely, samples n times, with replacement, from the precincts proportional to the weights of the precincts.

Usage

```
tri.sample(Z, n, seed = NULL, print.trail = FALSE, simplify = TRUE,
          return.precincts = TRUE, PID = "PID", known = "known")
tri.sample.stats(samp)
```

Arguments

`Z`	elec.data object
`n`	Either a audit.plan.tri object (that contains n) or an integer which is the size of the sample
`seed`	Seed to use.
`print.trail`	Print diagnostics and info on the selection process.
`simplify`	If TRUE, return a data frame of unique precincts sampled, with counts of how many times they were sampled. Otherwise return repeatedly sampled precincts seperately.
`return.precincts`	Return the precincts, or just the precint IDs
`PID`	The name of the column in Z\$V holding unique precinct IDs
`known`	Name of column in Z\$V of TRUE/FALSE, where TRUE are precincts that are considered "known", and thus should not be sampled for whatever reason.
`samp`	A sample, such as one returned from tri.sample

Details

The weights, if passed, are in the "e.max" column of Z\$V.

Value

tri.sample returns a sample of precincts. tri.sample.stats is a utility function returning the total number of unique precincts and ballots given a sample.

Author(s)

Luke W. Miratrix

See Also

trinomial.bound elec.data tri.calc.sample audit.plan.tri

Examples

```
data(santa.cruz)
Z = elec.data( santa.cruz, C.names=c("danner","leopold") )
samp = tri.calc.sample( Z, beta=0.75, guess.N = 10, p_d = 0.05,
               swing=10, power=0.9, bound="e.plus" )
tri.sample( Z, samp, seed=541227 )
```

---

trinomial.bound *Auditing with the Trinomial Bound: trinomial.bound and trinomial.audit*

Description

trinomial.audit converts the audited total counts for candidates to overstatements and taints. trinomial.bound calculates the trinomial bound given the size of an audit sample, the number of non-zero errors, and the size of the small-error threshold. It can also plot a contour of the distribution space, bounds, and alpha lines.

Usage

```
trinomial.audit(Z, audit)
trinomial.bound(n = 11, k = 2, d = 40, e.max = 100, xlim = c(0.4, 1),
       ylim = c(0, 0.55), alpha.lvls = c(10), zero.threshold = 0.3,
       tick.lines = NULL, alpha.lwd = 2, bold.first = FALSE, plot = TRUE,
       p.value.bound = NULL, grid.resolution = 300, ... )
```

Arguments

Z	An elec.data object that is the race being audited.
audit	A data.frame with a column for each candidate and a row for each audited precinct, holding the audit totals for each candidate. An additional column, count, holds the number of times that precinct was sampled (since sampling was done by replacement).
n	Size of the sample (not precincts, but samples which could potentially be multiple samples of the same precinct).
k	The number of positive taints found in sample.
d	The maximum size of a small taint. This is the threshold for being in the middle bin of the trinomial. All taints larger than d would be in the largest error bin.
e.max	The size of the largest error bin. Typically 100 (for percent) or 1.
xlim	Range of possible values of p0 worth considering
ylim	Range of possible values of pd worth considering

alpha.lvls	List of alphas for which bounds should be calculated. The first is the one that will be returned. The others will be graphed.
zero.threshold	Since the method calculates on a numerical grid, what difference between alpha and the calculated probabilty should be considered no difference.
tick.lines	A list of bounds. For these bound levels, add tick-lines (more faint lines) to graph
alpha.lwd	Line width for alpha line.
bold.first	TRUE/FALSE. Should first alpha line be in bold.
plot	Should a plot be generated.
p.value.bound	What is the bound (1/U) that would correspond to the entire margin. Finding the alpha corresponding to this bound is a method for finding the p-value for the trinomial bound test.
grid.resolution	How many divisions of the grid should there be? More gives greater accuracy in the resulting p-values and bounds.
...	Extra arguments passed to the plot command.

Details

Right now the p-value is computed in a clumsy, bad way. A grid of points over (0, xlim) X (0, ylim) is generated corresponding to values of p0 and pd, and for each point the mean of that distribution and the chance of generating an outcome as extreme as k is calculated. Then the set of points with an outcome close to alpha is extrated, and the corresponding bound is optimized over this subset. Not the best way to do things.

Value

List with charactaristics of the audit and the final results.

n	Size of sample.
k	Number of non-zero taints.
d	Threshold for what a small taint is.
e.max	The worst-case taint.
max	The upper confidence bound for the passed alpha-level.
p	A length three vector. The distribution (p0, pd, p1) that achieves the worst case.
p.value	The p.value for the test, if a specific worst-case bound 1/U was passed via p.value.bound.

Author(s)

Luke W. Miratrix

References

See Luke W. Miratrix and Philip B. Stark. (2009) Election Audits using a Trinomial Bound. http://www.stat.berkeley.edu/\~stark

See Also

See elec.data for information on the object that holds vote counts. See tri.sample for drawing the actual sample. See tri.calc.sample for figuring out how many samples to draw. See tri.audit.sim for simulating audits using this method. See CAST for an SRS audit method.

Examples

```
## Auditing Santa Cruz
data(santa.cruz.audit)
data(santa.cruz)
Z = elec.data(santa.cruz, C.names=c("leopold","danner"))
res = trinomial.audit( Z, santa.cruz.audit )
res
plan = tri.calc.sample( Z, beta=0.75, guess.N = 10, p_d = 0.05,
               swing=10, power=0.9, bound="e.plus" )

## Compute the bound.  Everything is scaled by 100 (i.e. to percents) for easier numbers.
trinomial.bound(n=res$n, k = res$k, d=100*plan$d, e.max=100, p.value.bound=100/plan$T,
           xlim=c(0.75,1), ylim=c(0.0,0.25),
           alpha.lvls=c(25), asp=1,
           main="Auditing Santa Cruz with Trinomial Bound" )
```

---

truth.looker *Looking at fake "truths" for election simulations*

Description

This prints out total error in a fake truth for an election, and some other info.

Usage

```
truth.looker(data)
```

Arguments

data	The data.frame returned from such things as make.audit

Details

Utility function for debugging and understanding stuff.

Look at a specific "truth" and print out what total error, etc. is.

Value

None. Just does printout.

weight.function *weight functions*

Description

This function produces weight functions to reweight found audit miscounts.

Usage

```
weight.function(name = c("no.weight", "weight",
             "weight.and.slop", "margin.weight",
             "taint"))
```

Arguments

name	name of function desired

Details

The functions are no weighting, weighted by size of precint, weight by size, after a slop of 2 votes has been taken off, and weighing for pairwise margin tests, and finally, the taint weight function that takes maximum error in precincts and gives a ratio of actual error to maximum error.

Value

A two-element list of two functions, the second being the inverse of the first. All the functions have three parameters, x, b_m, and M, which are the things to weight, the bound on votes (or maximum error in precincts), and the (smallest) margin.

Author(s)

Luke W. Miratrix

yolo *Yolo County, CA Election Data*

Description

This is for measure W in Yolo County, CA, November 2008. The file includes precinct-level reports.

Usage

```
data(yolo)
```

Format

A data frame with 114 observations on the following 8 variables.

`PID` Unique identifier for the batches of ballots

`Pct` The precinct id of the batch

`how` Vote by mail (VBM) or walk-in (PCT)

`b` Number of votes cast in that unit

`under` Number of undervotes (ballots not voted).

`over` Number of overvotes (where someone marked both yes and no).

`y` Reported number of valid ballots marked yes.

`n` Reported number of valid ballots marked no.

Details

In the actual audit, 6 precincts were selected (see example) and audited by hand-to-eye count by a group of 4 people cross-checking each other. One of the 6 batches had underreported the "yes" votes by 1, and one had overreported the "yes" votes by 1. There were no other errors.

Source

Yolo County, CA. Special thanks to Freddie Oakley and Tom Stanionis.

References

See Stark et al. for papers using this data to illustrate risk-limiting audits of election data.

Examples

```
# Make an elec.data object out of precicnt-level results
data(yolo)
yolo = make.Z( yolo, C.names=c("y","n","under","over"), tot.votes.col="b" )

# Look at different sample sizes and cuts for setting aside
# small precincts
CAST.calc.opt.cut( yolo, beta=0.75, stages=1, t=5, plot=TRUE )

print( yolo )

# Get details of the audit plan -- expected work, etc.
ap <- CAST.calc.sample( yolo, beta=0.75, stages=1, t=5, small.cut=5 )
print( ap )

# Draw a sample (seed not used for actual audit)
CAST.sample(yolo, ap, seed=12345678)
```

Index