

An assessment of the causes of the errors in the 2015 UK General Election opinion polls

Sturgis, P., Kuha, J., Baker, N., Callegaro, M., Fisher, S., Green, J., Jennings, W., Lauderdale, B. E., and Smith, P., *Journal of the Royal Statistical Society A*, 2017.

Supplementary materials

1. Introduction

The supplementary materials which are provided on a separate JRSS A/Wiley web site provide computer code, data and output to illustrate the calculation of point estimates of vote shares and measures of precision for them, obtained from data from election polls conducted as they were before the 2015 UK General Election. The following files are included in the materials:

- *ReadMe.txt*: List and explanation of the files.
- *PollingExample.R*: An R script for carrying out the analysis. Extracts and output from it are included in this note. It sources the contents of the next file.
- *PollingExampleDataAndFunctions.R*: Data (*PollData*) and three functions (*poll.estimation*, *poll.bootstrap*, and *quota.resample*) used in the example. Use the *source* function to load this file into R.

This note provides a more detailed explanation of how these files are used.

The example uses the statistical computing package R (see <https://cran.r-project.org/>). In addition to the functions provided in these supplementary materials, the add-on package *survey* should also be installed (T. Lumley (2004): Analysis of complex survey samples. *Journal of Statistical Software* 9(1): 1-19; T. Lumley (2014): "survey: analysis of complex survey samples". R package version 3.30; see *help(install.packages)* in R on how to install add-on packages).

2. Dataset

Analysis of the data in data frame *PollData* is used to illustrate the procedures. The data are based on, but not identical with, real poll data from the 2015 election. The data frame includes the following variables:

- Weighting variables (\mathbf{X} in the notation of the article):
 - *SexbyAge*: Sex and age group, cross-classified
 - *Region*: The region of Great Britain where the respondent lives
 - *PartyID*: Party identification, based on self-reported 2010 vote or other information
- Self-reported likelihood of voting, converted to a probability (T): *LTV*
- Intended vote (V): *VoteIntention*

Most polls in 2015 actually used a larger set of variables in the role of \mathbf{X} . The variables included here are, however, sufficient to illustrate the methods.

The dataset has $n=2000$ observations. The values and distributions of the variables in it are as follows:

\$SexbyAge

Male 18-24	Male 25-39	Male 40-59	Male 60+
81	172	384	314
Female 18-24	Female 25-39	Female 40-59	Female 60+
123	221	368	337

\$Region

North	Midlands	East	London	South	Wales	Scotland
425	292	182	204	432	206	259

\$PartyID

Lab	Con	LibDem	SNP or PC	Other	None or DK
624	514	173	74	130	485

\$LTV

0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
81	12	13	10	5	26	19	35	62	149	1588

\$VoteIntention

Con	Lab	LibDem	UKIP	Green	SNP	PC	Other	None or DK
614	528	182	190	86	128	16	29	227

In the example, we assume that *SexbyAge* and *Region* were used as quota variables in the quota sampling, and both they and *PartyID* are used as weighting variables in calibration (raking) weighting. The weighting targets (population distributions) for them are set as follows:

> sex.age.dist

	SexbyAge	Freq
1	Male 18-24	0.060
2	Male 25-39	0.126
3	Male 40-59	0.169
4	Male 60+	0.130
5	Female 18-24	0.059
6	Female 25-39	0.127
7	Female 40-59	0.173
8	Female 60+	0.156

> region.dist

	Region	Freq
1	North	0.246
2	Midlands	0.164
3	East	0.096
4	London	0.128
5	South	0.229
6	Wales	0.050
7	Scotland	0.087

> partyid.dist

	PartyID	Freq
1	Lab	0.316
2	Con	0.278
3	LibDem	0.095
4	SNP or PC	0.019
5	Other	0.049
6	None or DK	0.243

Here the numbers in the 'Freq' columns are target proportions.

3. Preparation for the analysis

For convenience of subsequent analysis, information on the poll and its design are collected together into a list, here called *Poll.information*:

```
Poll.information <- list(data=PollData,
  est.function=poll.estimation,
  quota.columns=quota.cols,
  weighting.margins=list(~SexbyAge,~Region,~PartyID),
  weighting.targets=list(sex.age.dist,region.dist,partyid.dist),
  LTV.variable="LTV",
  Vote.variable="VoteIntention",
  parties=levels(PollData$VoteIntention)[1:8]
)
```

The elements of this list are the data (*data*), a function for calculating poll estimates (*est.function*; this is discussed in section 4 below), list of dummy variables for categories of the quota variables (*quota.columns*), the weighting variables as a list of formulas (*weighting.margins*), the target distributions for the weighting variables (*weighting.targets*; see section 2), names of the variable for the assumed turnout probability (*LTV*) and the voting intention (*Vote.variable*) and a character vector (*parties*) which contains the labels of those categories of the voting intention variable which correspond to votes for a party (rather than “Don’t know” responses or stated intention not to vote). Here the value of *parties* is

```
> Poll.information$parties
[1] "Con"      "Lab"      "LibDem"   "UKIP"     "Green"    "SNP"      "PC"       "Other"
```

and the value of *quota.columns* is

```
> Poll.information$quota.columns
[1] "Dsexage1" "Dsexage2" "Dsexage3" "Dsexage4" "Dsexage5" "Dsexage6"
[7] "Dsexage7" "Dsexage8" "Dregion1"  "Dregion2" "Dregion3" "Dregion4"
[13] "Dregion5" "Dregion6" "Dregion7"
```

This is a set of dummy variables created as shown in the R script file.

For a specific poll, each of the elements of this list would be set to values which apply to that poll.

4. Point estimation

The function *poll.estimate* (which is here included as the *est.function* element of the list *Poll.information*) is used to calculate point estimates of vote shares (and the Conservative-Labour difference, all expressed as percentages). The function is listed on the next page. The estimation is carried out as described in Section 2.1 of the article. In this example, the results are

```
> Poll.information$est.function(Poll.information)

  Con-Lab      Con      Lab      LibDem      UKIP
6.5683324 36.8680890 30.2997567  10.8463919  9.8524024

  Green      SNP      PC      Other
5.5724562  4.6334781 0.3668932  1.5605326
```

Warning message:

```
In svydesign.default(ids = ~1, data = data) :
  No weights or probabilities supplied, assuming equal probability
```

(The warning message comes from a set-up step of the weighted estimation, and can be ignored.)

The estimation function assumes a single calibration weighting stage. If more complex weighting schemes are used, the function should be modified accordingly. For example, in 2015 some pollsters first carried out separate weighting for the sub-samples in (say) Scotland and the rest of Great Britain to target distributions for these countries, followed by final weighting for the population sizes across the countries.

```

> poll. estimation
function (poll.information)
{
# poll. estimation
#
# Supplementary materials for
# Sturgis, P., Kuha, J., Baker, N., Callegaro, M., Fisher, S., Green, J.,
# Jennings, W., Lauderdale, B. E., and Smith, P. (2017)
# `An assessment of the causes of the errors In the 2015
# UK General Election opinion polls'
#
# Function for calculating point estimates of cote shares
# Please see the article and an explanatory note in the supplementary
# materials for further information.
#
      require(survey)
#
      data <- poll.information$data
      weighting.margins <- poll.information$weighting.margins
      weighting.targets <- poll.information$weighting.targets
      LTV.variable <- poll.information$LTV.variable
      Vote.variable <- poll.information$Vote.variable
      parties <- poll.information$parties
#
      svy.unweighted <- svydesign(ids=~1,data=data)
#
# Calibration (raking) weighting:
#
      svy.weighted <- rake(svy.unweighted,
                          sample.margins = weighting.margins,
                          population.margins = weighting.targets
                          )

svy.weighted$variables<- data.frame(svy.weighted$variables,
      Wraked=weights(svy.weighted))
svy.weighted$variables$Wnew <-
svy.weighted$variables$Wraked*svy.weighted$variables[,LTV.variable]
      data.tmp <- svy.weighted$variables
      data.tmp <- data.tmp[data.tmp[,Vote.variable] %in% parties,]
      svy.weighted <- svydesign(~1,data=data.tmp,weights=data.tmp$Wnew)
#
# Weighted point estimates
#
      res.tmp <- prop.table(svytable(as.formula(paste("~",Vote.variable)),
      svy.weighted))
#
      res <- rep(NA,length(parties)+1)
      names(res) <- c("Con-Lab",parties)
      res[-1] <- res.tmp[names(res.tmp)%in% parties]
#
# Conservative-Labour difference
## This assumes that the labels for Conservatives and Labour in
## Vote.variable include the strings "Con" and "Lab":
#
      res[1] <- res[pmatch("Con",names(res))]-res[pmatch("Lab",names(res))]
```

```

      return(100*res)
}

```

5. Bootstrap estimation of sampling variability in the estimates

Estimation of sampling variability in the poll estimates is done using a bootstrap resampling procedure, as described in Section 2.2 of the article. The estimation is done by the function *poll.bootstrap*, as described below. Inside this function, bootstrap resampling is done by the function *quota.resample*. As described in the article, it draws samples in such a way that the distribution of the quota variables (i.e. the totals of the dummy variables listed in the *quota.columns* element of the list which describes the poll sample, as defined in section 3 above) in the resample should match those in the observed data (unless the algorithm stops early with some quotas not quite filled, because it ran out of observations which match all the remaining unfilled quotas; this did not happen in the illustrative example considered here).

The arguments of the *poll.bootstrap* function are as follows:

```
> args(poll.bootstrap)
function (poll.information, n.bootstrap = 0,
        bootstrap.samples = NULL, alpha = 0.95, bca = F)
```

where

- *poll.information* is a list which describes the poll sample, as defined in Section 3 above
- *n.bootstrap* is the number of bootstrap samples to be drawn
- *bootstrap.samples* is a list returned by a previous call to *poll.bootstrap*. If it is not NULL, no new bootstrap samples are drawn, and the function only calculates the confidence intervals
- *alpha* is the confidence level of the confidence intervals
- *bca*: if this is TRUE (T), adjusted percentile (BCA) confidence intervals should also be calculated; if it is FALSE, only standard normal and standard percentile intervals are calculated

If *n.bootstrap=0* and *bootstrap.samples=NULL*, the function only calculates the point estimates from the actual data. If *n.bootstrap* is greater than 0 and *bootstrap.samples=NULL*, *n.bootstrap* new bootstrap samples are drawn and then used to calculate bootstrap estimates of standard errors and confidence intervals. If *bootstrap.samples* is not NULL, no new bootstrap samples are drawn but the results in *bootstrap.samples* are used to calculate bootstrap estimates of standard errors and confidence intervals

Three kinds of bootstrap confidence intervals are reported: standard normal intervals, standard percentile intervals, and (if *bca=T*) adjusted percentile (BCA) intervals. Denoting by *p* the estimate of a vote share (or share difference) calculated from the actual data, and α the confidence level, the standard normal interval is $p \pm q(1-[1-\alpha]/2)se$ where $q(1-[1-\alpha]/2)$ is the $1-[1-\alpha]/2$ quantile of the standard normal distribution (e.g. 1.96 for $\alpha=0.95$) and *se* is the standard deviation in the estimates of the vote share across the bootstrap samples. The standard percentile interval is between the $([1-\alpha]/2)*100\%$ and $(1-[1-\alpha]/2)*100\%$ percentiles (e.g. 2.5% and 97.5% for $\alpha=0.95$) of the estimates across the bootstrap samples. The BCA intervals have been calculated using the simple method described in Chapter 14 of Efron and Tibshirani (*An Introduction to the Bootstrap*, 1993) with code adapted from the *bcanon* function in the R package *bootstrap* by Rob Tibshirani.

With an old laptop, drawing 10,000 bootstrap samples for this example (with its dataset of size $n=2000$) took around 28 minutes, and calculating the BCA intervals an additional 2 minutes. Results from running this procedure for the example are shown below. For example, for the difference in vote shares between Conservatives and Labour, the point estimate from the actual data is +6.57 percentage points. Bootstrap estimate of the standard error of this

estimate is 1.57, and 95% standard normal, standard percentile and adjusted percentile intervals are (3.48; 9.65), (3.52; 9.75) and (3.51; 9.73) respectively. The adjusted percentile intervals are preferred on general theoretical grounds, but here all the intervals are similar (and identical to one decimal place).

```
> Poll.bootstrap.res <- poll.bootstrap(Poll.information,n.bootstrap=10000)
> Poll.bootstrap.res <- poll.bootstrap(Poll.information,
    bootstrap.samples=Poll.bootstrap.res,bca=T)
```

```
> Poll.bootstrap.res[1:2]
```

```
$summary
```

	Actual est.	BS Mean	BS Median	BS se
Con-Lab	6.568	6.593	6.573	1.573
Con	36.868	36.883	36.874	0.952
Lab	30.300	30.290	30.296	0.929
LibDem	10.846	10.859	10.854	0.713
UKIP	9.852	9.851	9.840	0.689
Green	5.572	5.554	5.547	0.584
SNP	4.633	4.634	4.633	0.293
PC	0.367	0.368	0.364	0.093
Other	1.561	1.561	1.552	0.301

```
$conf.intervals
```

	Normal.L95	Normal.U95	Percentile.L95	Percentile.U95	BCA.L95	BCA.U95
Con-Lab	3.484	9.652	3.523	9.749	3.513	9.729
Con	35.003	38.733	35.029	38.773	35.008	38.753
Lab	28.479	32.121	28.478	32.115	28.482	32.122
LibDem	9.448	12.244	9.467	12.273	9.453	12.246
UKIP	8.501	11.203	8.531	11.221	8.542	11.232
Green	4.428	6.716	4.441	6.709	4.493	6.775
SNP	4.059	5.207	4.054	5.210	4.051	5.207
PC	0.185	0.549	0.199	0.565	0.202	0.571
Other	0.972	2.150	1.005	2.181	1.015	2.198

First 6 of the 10000 sets of estimates from the 10000 bootstrap samples:

```
> round(head(Poll.bootstrap.res[[3]]),3)
```

	Con-Lab	Con	Lab	LibDem	UKIP	Green	SNP	PC	Other	N
[1,]	2.553	34.977	32.423	10.542	9.369	6.108	4.640	0.383	1.558	2000
[2,]	9.737	38.545	28.808	11.036	9.906	5.258	4.606	0.451	1.390	2000
[3,]	5.798	36.807	31.010	11.537	9.091	5.201	4.773	0.121	1.459	2000
[4,]	7.642	37.213	29.571	11.154	9.278	6.754	4.281	0.231	1.517	2000
[5,]	7.828	37.425	29.597	11.332	9.729	5.024	4.867	0.414	1.612	2000
[6,]	6.832	37.591	30.759	10.823	9.307	5.109	4.359	0.381	1.672	2000