An interesting big data thought experiment

The other day on Twitter I saw someone referencing a paper or a seminar or something that was reported to examine the following situation: if you have an urn with a million balls in it of two colours (say red and white) and you want to estimate the proportion of balls that are red, are you better off taking the top 800,000 balls – or stirring the urn and taking a sample of just 10,000? The answer given was the second of these options. The idea is to illustrate the limitations of "big data" methods, which can often be taken to mean samples that are very large but of uncertain quality with regard to representativeness and randomness.

The nature of the Twitter user experience is such that I've since lost track of the original post and can't find it again.

My first thought was "wow, what a great illustration!" My second was "actually, that sounds a bit extreme." After all, worsening your estimate by 80:1 a pretty severe design effect penalty to pay for non-random sampling. A bit of thinking shows that whether the small, random sample outperforms the bigger one is going to depend very much on how the urn was filled in the first place.

Consider as an extreme example that the urn was filled by pouring in all the balls of one colour, then all of another. In this situation you will certainly be better off with the stirring method (in all that follows, I am going to assume that the stirring is highly effective, and that stirring and sampling equates to taking a simple random sample).

But at the other extreme, if the urn was filled in a completely random order, than either sampling method equates to simple random sampling, and the larger sample will greatly outperform the smaller. In fact, in that case the standard errors from the stirring method will be 8.9 times from the large data method (square root of 80). So the choice between the methods depends on how spatially correlated (in 3 dimensional space) the colour of the balls is within the urn. This is similar to how the need for special time series methods depends on whether there is autocorrelation over time; spatial methods depends on whether there is spatial autocorrelation; and adjusting for survey design effects depends on intra-cluster correlation.

Efficiently simulating filling an urn with balls

To explore just how correlated the balls need to be for the simple random sampling method to be preferred, I ran a simple simulation. In this simulation, the balls are added to the urn one at a time, with no natural stirring. There is an overall probability `p` of a ball being red (and this is the exact probability of the first ball that goes in being red), but for the second and subsequent balls there is a second probability to be aware of, `q`, which is the chance that this ball will be forced to be *the same colour as the previous ball*, rather than pulled from the hyper population with parameter `p`.

Here's a simple R function that will generate an urn of size `n` with those parameters:

Post continues after R code.

```
library(tidyverse)
library(scales)
library(Rcpp)
library(microbenchmark)


#' Simulate filling of an urn with red and white balls, with serial
correlation
#'
#' @param n Number of red and white balls in the urn
#' @param p overall hyper-population proportion of red balls
#' @param q probability that a ball, when originally placed in the urn
in order, will be the same
#' colour as the preceding ball rather than randomly chosing between
red and white with probability p
```

```
fill_urn_r <- function(n, p, q){
  x <- numeric(n)
  x[1] <- rbinom(1, 1, prob = p)

  for(i in 2:n){
    x[i] <- ifelse(rbinom(1, 1, prob = q) == 1,
                   x[i-1],
                   rbinom(1, 1, prob = p))
  }
  return(x)
}
```

However, this R program is too slow for my purposes. I'm going to want to generate many thousands of these urns, with a million balls in each, so time really matters. It was worth re-writing this in C++ via the Rcpp R package.

Post continues after R code.

```
# C++ version, hopefully much faster:
cppFunction('NumericVector fill_urn_c(int n, double p, double q) {
  NumericVector x(n);
  x[0] = as(rbinom(1, 1, p));
  for (int i=1; i(rbinom(1, 1, q)) == 1){
    x[i] = x[i-1];
   } else {
    x[i] = as(rbinom(1, 1, p));
   }
  }

  return x;
}')

# Check the R and C++ versions give same results:
all.equal({set.seed(123); fill_urn_c(30, 0.3, 0.9)},
          {set.seed(123); fill_urn_r(30, 0.3, 0.9)})

# Compare speeds
microbenchmark("C++" = fill_urn_c(10000, 0.3, 0.9),
               "R" =   fill_urn_r(10000, 0.3, 0.9))
```

The C++ function was as easy to write as the R version (helped by the Rcpp function `rbinom` which makes C++ seem even more familiar to R users) and delivers roughly a 40- or 50-fold speed up. Here's the results of that benchmarking at the end of the last chunk of code:

```
Unit: microseconds
 expr     min       lq      mean   median       uq      max neval
  C++   705.9   755.85   991.928   851.15   960.3   6108.1   100
    R 31868.7 36086.60 69251.103 41174.25 48143.8 427038.0   100
```

By the way, the `as()` in my Rcpp code is needed because `rbinom` generates a vector (of size one in this case) and I want to treat it as a scalar. There may be a more elegant way of handling this, but this was good enough for my purposes.

Comparing the two sampling methods

To help with comparing the two sampling methods, I write two more functions:

- `compare_methods()` to run the "big data" and random sampling methods on a single urn and compare their results to the true proportion in that particular urn (using the actual proportion in the urn,

not the hypothetical hyper-parameter p)

- `overall_results()` to generate many urns with the same set of parameters

Post continues after R code.

```
#------------------Comparing two sampling methods-----------

#' Compare two methods of sampling
#'
#' @param x a vector of 1 and 0 for which we want to estimate the
proportion of 1s
#' @return a data frame with one row and two columns, showing the
difference between two
#' methods of sampling and estimating a proportion
#' @details The "big data" method takes the most recent 80% as the
sample. The "sample" method
#' takes a simple random sample.
compare_methods <- function(x){
  n <- length(x)

  correct <- mean(x)

  # Method 1 - the most recent 80% of observations:
  bigdata_method <- mean(tail(x, round(n * 0.8))) - correct

  # Method 2 - simple random sample of 1% of observations:
  sample_method <- mean(sample(x, round(n * 0.01))) - correct

  return(data.frame(
    bigdata_method = bigdata_method,
    sample_method = sample_method
  ))
}

#' Repeated comparison of two methods of sampling
#'
#' @param m Number of times to run the experiment
#' @param n Number of red and white balls in the urn for which we are
trying to estimate a proportion
#' @param p overall hyper-population proportion of red balls
#' @param q probability that a ball, when originally placed in the urn
in order, will be the same
#' colour as the preceding ball rather than randomly chosing between
red and white with probability p
#' @details This works by repeated calls to fill_urn_c() and
compare_methods()
overall_results <- function(m, n, p, q){
  results <- lapply(1:m, function(j){
    x <- fill_urn_c(n, p, q)
    y <- compare_methods(x)
    return(y)
  })

  tmp <- do.call(rbind, results) %>%
    tidyr::gather(variable, value) %>%
    dplyr::mutate(m = m,
                  n = n,
                  p = p,
```

```
                    q = q)
  return(tmp)
}
```

Here's what I get when I compare the two methods for a thousand runs with urns of 10,000 balls each, with p = 0.3 and various values of q:

… and here are the results for the original use case, a thousand runs (at each value of q) of an urn with one million balls:

So we can see in both cases we need a *lot* of serial correlation between balls (based on the order they go into the urn) for the method of random sampling 1% of the balls to out-perform the brute force selection of the top 80% of balls. Somewhere between a value for q of 0.99 and 0.999 is when the stirring method is clearly better. Remember, q is the probability that any ball going into the urn is the same as the previous ball, before the alternative colour selection being chosen with probability p (0.3 in our case).

Here's the code for the actual simulations.

Post continues after R code.

```
#--------------------------------Small urns--------------
tmp1 <- overall_results(1000, 1e4, 0.3, 0.9)
tmp2 <- overall_results(1000, 1e4, 0.3, 0.99)
tmp3 <- overall_results(1000, 1e4, 0.3, 0.999)
tmp4 <- overall_results(1000, 1e4, 0.3, 0.9999)

rbind(tmp1, tmp2, tmp3, tmp4)  %>%
  mutate(variable = case_when(
    variable == "bigdata_method" ~ "Sample 80% of balls at top of
urn",
    variable == "sample_method" ~ "Stir urn and take random sample of
1%"
  )) %>%
  mutate(q = paste(q, "serial relationship factor")) %>%
  ggplot(aes(x = value, fill = variable, colour = variable)) +
  geom_density(alpha = 0.5) +
  facet_wrap(~q, scales = "free") +
  labs(fill = "Sampling method:", colour = "Sampling method:",
       x = "Discrepency between sample proportion and true proportion
from entire urn",
       title = "Comparison of 'big data' and random sampling methods",
       subtitle = "Estimating a proportion from an urn of 10,000 red
and white balls.
Which method works best depends on the degree of serial correlation
between balls.",
       caption = "http://freerangestats.info")

#----------------------Large urns------------------
tmp5 <- overall_results(1000, 1e6, 0.3, 0.9)
tmp6 <- overall_results(1000, 1e6, 0.3, 0.99)
tmp7 <- overall_results(1000, 1e6, 0.3, 0.999)
tmp8 <- overall_results(1000, 1e6, 0.3, 0.9999)

rbind(tmp5, tmp6, tmp7, tmp8)  %>%
  mutate(variable = case_when(
    variable == "bigdata_method" ~ "Sample 80% of balls at top of
urn",
    variable == "sample_method" ~ "Stir urn and take random sample of
1%"
```

```
  )) %>%
  mutate(q = paste(q, "serial relationship factor")) %>%
  ggplot(aes(x = value, fill = variable, colour = variable)) +
  geom_density(alpha = 0.5) +
  facet_wrap(~q, scales = "free") +
  labs(fill = "Sampling method:", colour = "Sampling method:",
       x = "Discrepency between sample proportion and true proportion
from entire urn",
       title = "Comparison of 'big data' and random sampling methods",
       subtitle = "Estimating a proportion from an urn of 10,000 red
and white balls.
Which method works best depends on the degree of serial correlation
between balls.",
       caption = "http://freerangestats.info")
```

One final point – what if we look at the absolute value of the discrepency between each methods estimate of the proportion and its true value. We see basically the same picture.

```
rbind(tmp5, tmp6, tmp7, tmp8)  %>%
  mutate(variable = case_when(
    variable == "bigdata_method" ~ "Sample 80% of balls at top of
urn",
    variable == "sample_method" ~ "Stir urn and take random sample of
1%"
  )) %>%
  mutate(q = paste(q, "serial relationship factor")) %>%
  mutate(value = abs(value)) %>%
  ggplot(aes(x = value, fill = variable, colour = variable)) +
  geom_density(alpha = 0.5) +
  facet_wrap(~q, scales = "free") +
  labs(fill = "Sampling method:", colour = "Sampling method:",
       x = "Absolute value of discrepency between sample proportion
and true proportion from entire urn",
       title = "Comparison of 'big data' and random sampling methods,
urn with 1,000,000",
       subtitle = "Estimating a proportion from an urn of 10,000 red
and white balls.
Which method works best depends on the degree of serial correlation
between balls.",
       caption = "http://freerangestats.info")
```

Reflections

If we thought the data generating process (ie how the urn was filled with balls) resembled my simulation, you would be better off choosing the large sample method (assuming equal cost) unless you had reason to believe the serial relationship factor `q` was very high. But this doesn't invalidate the basic idea of the usefulness of random sampling. This is for several reasons:

- The costs are unlikely to be the same. Even with today's computers, it is easier and cheaper to collect, process and analyse smaller than larger datasets.
- The "big data" method is *risky*, in the sense that it makes the analyst vulnerable to the true data generating process in a way that simple random sampling doesn't. With random sampling we can calculate the properties of the statistic exactly and with confidence, so long as our stirring is good. We can't say the same for the "top 80%" method.
- Related to the point above, the risk is particularly strong if the data generating process is quirkier than my simulation. For example, given my simulated data generating process, both sampling methods produce unbiased results. However, this isn't always going to be the case. Consider if the urn had been filled on the basis of "put all the red balls in first; then fill up the rest of the urn with white balls". In

this case the “top 80%” method will be very badly biased to underestimate the proportion of red balls (in fact, with p = 0.3, the method will estimate it to be 0.125 – a ghastly result).

That final dot point might sound perverse, but actually it isn’t hard to imagine a real life situation in which data is generated in a way that is comparable to that method. For example, I might have one million observations of the level of sunlight, taken between 1am and 9am on a November morning in a temperate zone.

So my overall conclusion – a small random sample will give better results than a much larger non-random sample, under certain conditions; but more importantly, it is reliable and controls for risk.