Reservior Sampling

Al-waili, Ahmed *aalw001@gold.ac.uk*

February 26, 2018

What is Reservior Sampling?

A unbiased sampling that is taken place in a data stream with an unknown length.

What does that exacly mean?

Well, lets say we have a stream of continous data, that just keeps coming in. We need to sample this data for research, so what we do we assign a maximum value we could have to this data lets call it *i*. Then we apply to this maximum value a sample size, lets call this an array/list of *s*. What we then do is apply the reservoir sampling to this stream of data, and this allows us to output a unbias set of data from the stream.

How does it work?

First we must understand the goal, and our goal is to output an item from the stream randomly with the probability must be equally likely, such that

$$P_{ai} = \frac{1}{n} \tag{1}$$

Proof of this shown below:

$$\begin{aligned} P_{s_i=a_i} &= \frac{1}{i}(1 - \frac{1}{i+1}) * (1 - \frac{1}{i+2})...(1 - \frac{1}{n}) \\ &= \frac{1}{i}(\frac{i+1-1}{i+1}) * (\frac{i+2-1}{i+2})...(\frac{n-1}{n}) \\ &= \frac{1}{n} \end{aligned} \tag{2}$$

Now we have proven that we can randomly take out from stream with $1/n$ we can look on how to sample the data. So, lets have a look on how to sample data with k items from a stream *s*. Lets make the max size we evaluate *aj*, and the samples *sj*. Now instead of holding one memory cell of what the sample we want, we hold k number of sample cells, and for the first k items (sk) we simply assign ai up to sj. Now we generate a random number from 1 to *i* (Uniform at random) If the random number happens to be 1 and k, we replace the memory cell contents of sj with ai.

We then do this the number of times we want to sample (length of sj).

Real life example of Reservoir Sampling

Reservoir has many real life applications, in major industries such as Marketing and Search Engines. The reason behind Marketing is for advterisments

as what happens during the night is completly different to what happens in the day, and this is therefore a continious stream of data. Therefore, sampling is done for these data streams and advertisments are shown to users. Now this may be more complicated than it needs to be so, to make you understand lets us a famous analogy of a Dating Show.

Now imagine a dating show, where the bachelorette is seated beside an empty chair (the samples). The host proceeds to introduce a suitor into the empty seat, next the host invites a second suitor. Now the bachelorette is given a choice between the two suitors, by asking questions or decision making. Now the host, introduces a third suitor in the same way now the bachelorette can decide to replace or keep one of the two suitors. After showing *n* suitors this way, the game show ends and the bachelorette choses the suitor. However, if the bachelorette decides to choose a suitor by flipping a coin, this will be unfair, as this therefore needs to be an *n* sided coin and equal probablity for each side (each suitor to be chosen). Now, for the suitor to be chosen they must survive *n-1* coin flips. The bachelorette needs to give a fair change 1/n to all suitors, and since the last suitor only wins if and only if the bachelorette decides to swap on the last step, and for the second to last suitor to win the bachelorette must decide to swap on the (n-1)th step. This outcomes to

$$P_(n-1)(n-1)/n \tag{3}$$

Where P(n-1) is the probablity that the bachelorette swaps on the (n-1)th steps (n-1)/n where she does not decide to swap, this solves for:

$$P_{(n-1)}(n-1)/n = 1/n \tag{4}$$

Time complexity of Reservior Sampling -

$$O(n) \tag{5}$$

https://dl.acm.org/citation.cfm?id=198435

```
Class reservoir(object):
        def __init__(self, max_size):
            self.samples = []
            self.max_size = max_size
            self.i = 0

        def add(self, element, timestamp):
            size=len(self.max_size)
            if size >= self.max_size:
```

```
 spot=random.randint(0, self.i - 1)
if spot < size:
    self.samples[spot]=(element, timestamp)
else:
    self.samples.append((element, timestamp))

self.i += 1
```