Bayesian update for bin model to illustrate computation catalyst idea

Consider a bin with two kinds of ball, red and green

Suppose we have two possible bins from A and B. If the bin is A, the probability is given

P(red |H=A) =0.6

P(green |H=A) =0.4

If the bin is B, the probability is given by

P(red |H=B) = 0.3

P(green |H=B) = 0.7

The prior probability is ***assumed*** to be

P(H=A) =1/2

P(H=B)= 1/2

Then we draw a ball suppose the ball red, then we calculate the posterior probability

$$P(H=A|\text{red}) = \frac{P(red|\text{H=A})P(H=A)}{P(red)}$$

Note that we can calculate P(red) by

P(red) = P(red | H=A)P(H=A)+ P(red | H=B)P(H=B)

$$P(H=B|\text{red}) = \frac{P(red|\text{H=B})P(H=B)}{P(red)}$$

Now once the posterior probability is calculated, we set the new prior probability

P(H=A)= P(H=A | red)

P(H=B)= P(H=B | red)

Similarly, the ball is green, we update the posterior probability

$$P(H=A|\text{green}) = \frac{P(green|\text{H=A})P(H=A)}{P(green)}$$

$$P(H=B|\text{green}) = \frac{P(green|\text{H=B})P(H=B)}{P(green)}$$

with

P(H=A | green) = P(| H=A) P(H=A) /(P(white | H=A)P(H=A)+ P(white | H=B)P(H=B))

we set the new prior probability for the next update according to the color of the current ball(data)

P(H=A)= P(H=A | green)

P(H=B)= P(H=B | green)

Let's test a case. We pick bin to be A so a randomized experiment is executed so the true probability

P(H=A) =1

P(H=B)= 0

This is unknown to the agent who conduct the experiment so he or she assume the initial prior probability to be

P(H=A) = 0.5

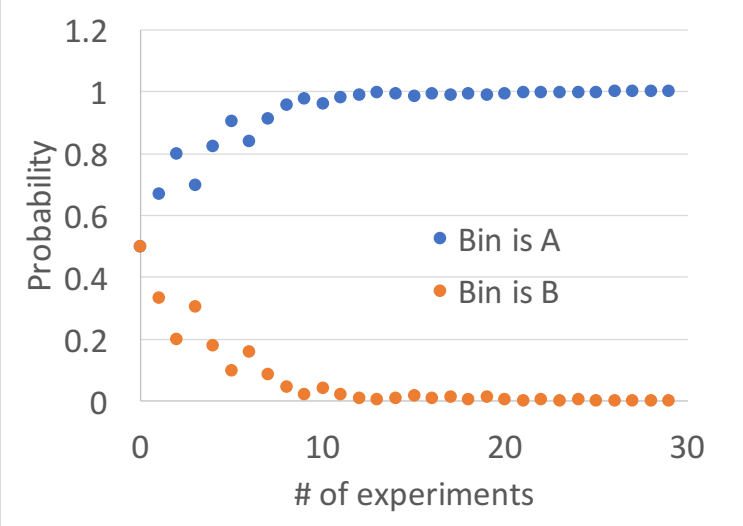
P(H=B)= 0.5

This is of course not correct and just a random guess. We implement the algorithm in python and obtain the updated probability for 30 experiments.

So after 30 experiments, the new prior probability is

P(H=A)= 0.9991

This is very close to 1 so we can see the Bayesian updating algorithm is giving us the correct answer given that we conduct a large number of experiments.



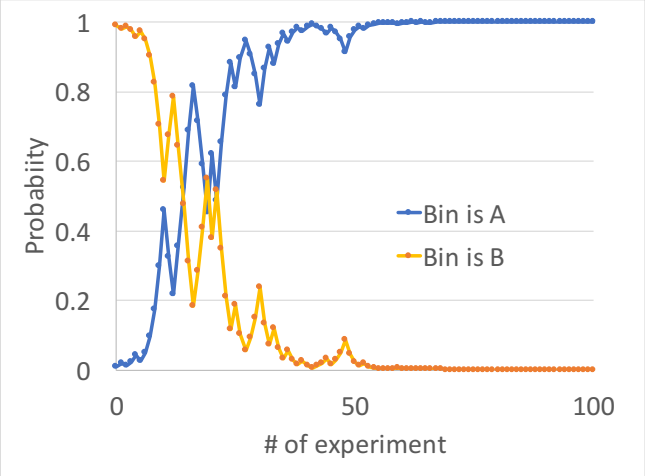
How about a more radical and irrational guess that prior probability of A is only 0.01?

P(H=A) = 0.01

P(H=B)=0.99

This is farm from the truth. We assume the probability is 1% as initial guess.

We run 100 experiments and obtain the result shown below



The Bayesian update gives us the correct result at the expense of 100 trials. So it illustrate the point that initial prior probability does not matter as long as we have enough “data” or trials to update the probability.

To summarize, we wish to make these points:

- Bayesian probability formalism automatically make a learning algorithm.
- We first model the system by making assumption on the prior and use Bayesian formula to update our current belief.
- The process is iterated as new evidence is generated from our experiment.
- We establish this algorithm indeed works regardless which initial prior we start with.
- We can consider prior is “computational catalyst” to get the computation going.

Reference:

1. まなびのずかん統計学の図鑑，涌井良幸, 涌井貞美

Appendix: A note on the python coding:

This code only uses a library called “random”.

We ask the program to generate a random number X from uniform distribution form the interval [0,1]. If X> p, we assign the ball to be red. Otherwise, the ball is green.

```python
import random

#initial prior probabity
priorA = 0.01    #prior probability
priorB = 0.99    # prior proability

Pgreen_A= 4/10   # Probability of green given Bin A
Pred_A=6/10      # Probability of red given Bin A
Pgreen_B= 7/10   # Probability of green given Bin B
Pred_B= 3/10     # Probability of red given Bin B
#define these probabity and set to zero for convenience
P_A_red=0
P_A_green=0
P_B_red=0
P_B_green=0
num_seq=100

for seq in range(num_seq):
    x=random.uniform(0,1)

    if x>=0 and x<=Pgreen_A:
        P_A_green= Pgreen_A*priorA/(Pgreen_A*priorA+Pgreen_B*priorB)
        P_B_green=1-P_A_green
        priorA=P_A_green
        priorB=P_B_green
        print( 'green', seq+1, round(P_A_green,4))

    else:
        P_A_red= Pred_A*priorA/(Pred_A*priorA+Pred_B*priorB)
        P_B_red=1-P_A_red
        priorA=P_A_red
        priorB=P_B_red
        print( 'red', seq+1, round(P_A_red,4))
```