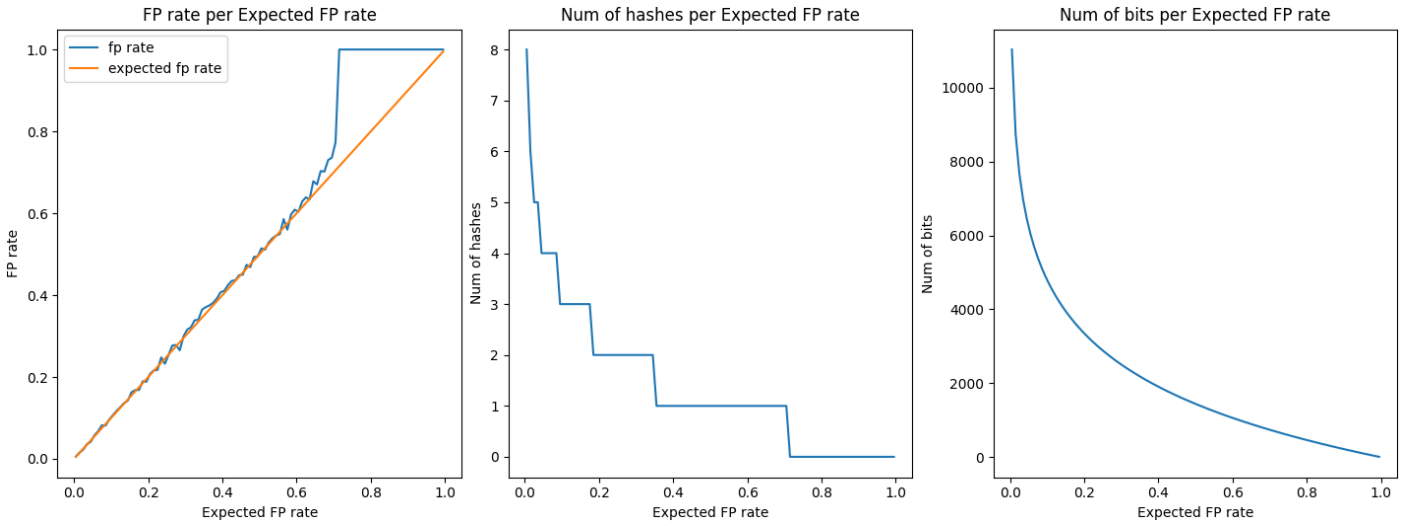
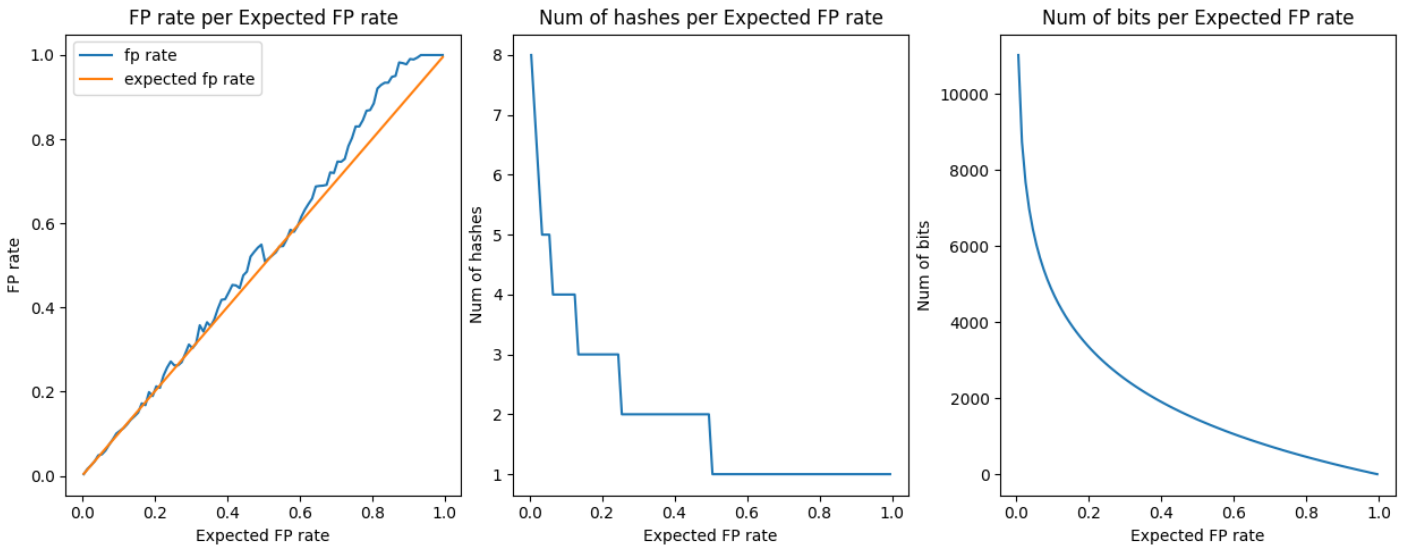
Bloom Filter

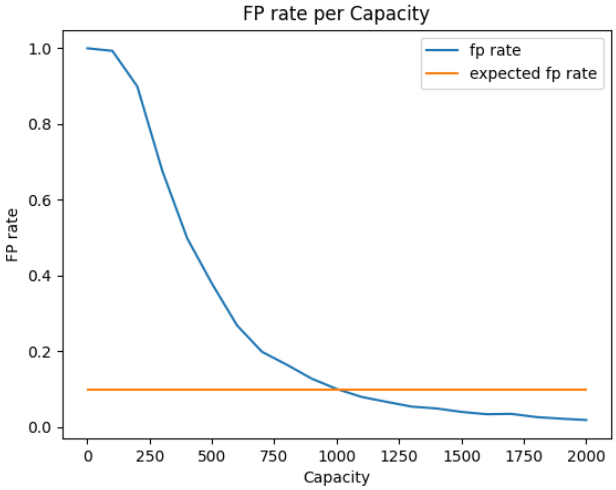


[그림 1]



[그림 2]

[그림 1]과 [그림 2]는 capacity = 1000인 BloomFilter에 '1'에서 '1000'까지 put() 해주었다. BloomFilter의 fp_prob 인자값을 0.005부터 1까지 0.01씩 증가시켜가며 나온 false positive 비율의 양상을 살펴보았다. Expected fp rate은 fp_prob으로 준 인자값을 나타내고, fp rate은 실제로 구한 false positive 비율이다. 이 둘의 양상이 비슷한 것으로 보아 false positive 비율이 의도한 대로 잘 나온 것 같다. [그림 1]에서는 expected fp rate이 0.7 부근일 때부터 fp rate이 1이 되는 것을 볼 수 있다. 이는 이전에 최적의 hash 함수의 개수를 구할 때 round() 함수를 써서 발생한 현상으로, math.ceil() 함수로 바꿔주어 [그림 2]와 같은 결과를 얻을 수 있었다. 하지만, round() 함수를 이용하여 최적의 hash 함수의 개수를 구했을 때가 hash 함수가 0개가 되지 않는 이상 더욱 정확하게 false positive 비율을 구함을 알 수 있다.



[그림 3]

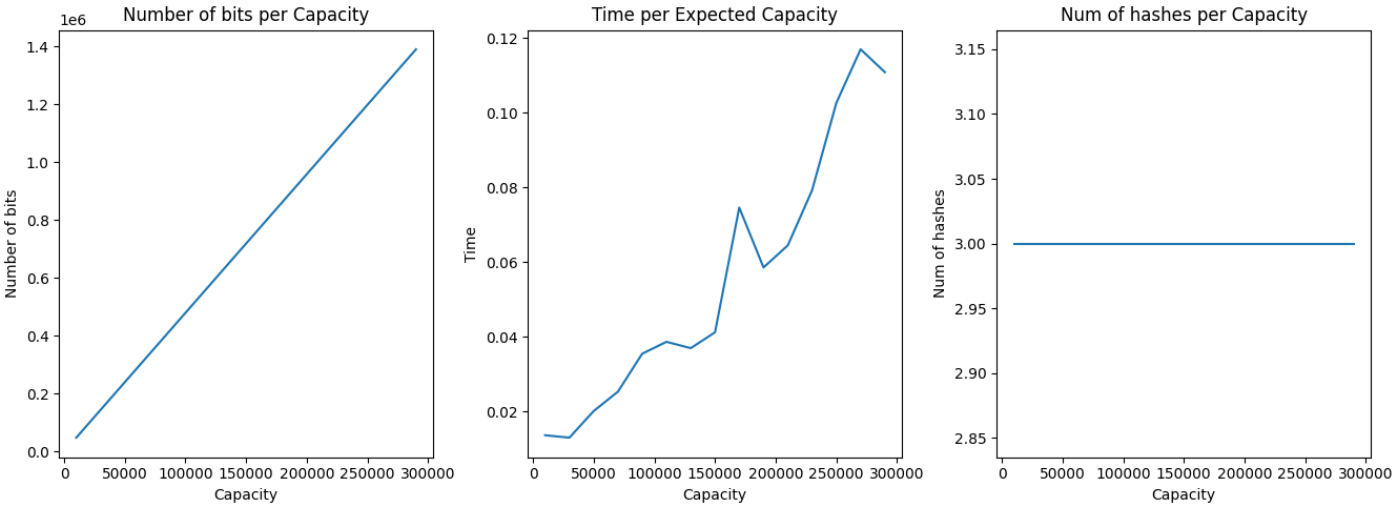
[그림 3]은 BloomFilter의 fp_prob 값을 0.1로 고정하고, capacity를 1부터 2000까지 100씩 증가시켜가며 구한 false positive 비율이다. capaciity가 작을수록 낮은 정확도를 보이고, 1000 부근에서 가장 높은 정확도를 보인다.

Q. 1억명의 사용자 계정이 시스템에 저장되어있고, 사용자가 회원가입 중에 동일한 계정명이 서버에 존재하는지 즉각 확인해주는 시스템을 개발하려고 합니다. 이 때, Bloom Filter를 어떻게 활용하면 좋을까요? 비트배열의 크기, false negative 값 등은 어떻게 설정하면 적절할까요? 생각을 서술해보세요.

$$\left(\frac{1}{2}\right)^{k} = \left(\frac{1}{2}\right)^{\frac{n}{m} ln(2)}$$

[그림 4]

[그림 4]는 k가 최적일 때 false positive 비율을 나타낸다. 이를 통해 k값이 증가할수록 false positive 비율이 감소함을 알 수 있다. 문제에서 1억명의 사용자 계정이 시스템에 저장되어 있다고 하였으므로, capacity = m = 1억이다. 그렇다면 n(비트배열의 크기)이 커지면 커질수록 false positive 값이 작아지므로 좋다고 생각할 수도 있다. 하지만, 즉각적으로 확인을 해주는 시스템을 고려해야 하므로 지연시간이 길면 안 된다.



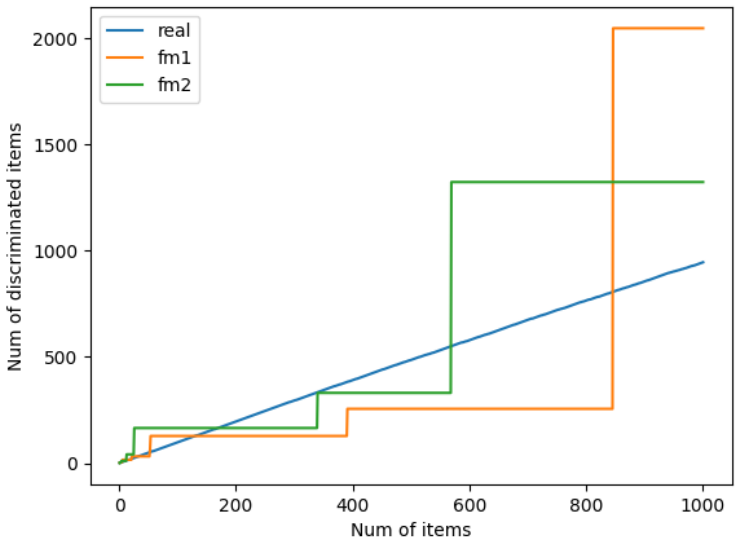
[그림 5]

[그림 5]는 fp_prob 값을 고정하고, capacity 값에 변화를 주었을 때 비트의 개수, test 하는데 걸리는 시간, hash 함수의 개수의 변화를 나타낸 것이다. 이를 통해 capacity가 커지면, bit수가 늘어나고, 이에 따라 test 하는데 걸리는 시간도 증가하는 양상을 보임을 알 수 있다. 따라서 false positive 값이 1/8 정도 되도록 하는 비트 배열의 크기로 설정하면 적절할 것 같다.

Q. Bloom Filter를 사용하면 좋을만한 상황을 얘기하고, 왜 그러한 상황에서 Bloom Filter가 도움이 되는지 설명해보세요.

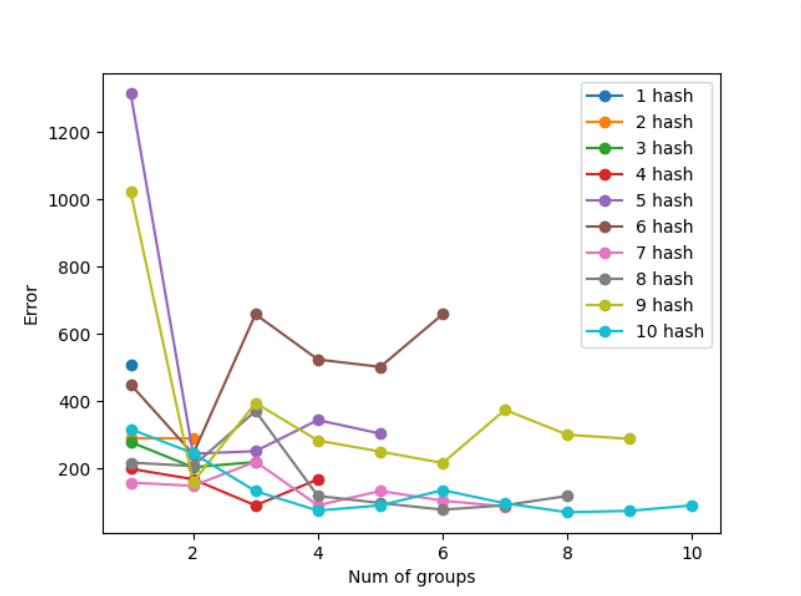
스팸 메일인지 확인할 때 좋을 것 같습니다. 스팸 메일은 번호에 비해 수도 많고, 아주 정확한 스팸 분류 결과를 요구하지 않고, 빠르게 검사가 이루어져야 하기 때문에 BloomFilter로 미리 스팸 메일을 저장해두고, 메일이 올 때마다 확인하면 효율적으로 검사할 수 있을 것 같습니다.

Flajolet-Martin



[그림 6]

RMSE를 이용하여 version 1과 version 2의 오차를 구해본 결과 대부분의 경우 version 2에서의 오차값이 더 작았다. Version 1의 오차가 더 작은 경우(더 정확한 경우)도 있었지만, version 1에서 r(a)값이 정말 크게 나오는 경우가 있어서 version 2가 더욱 안정적인 결과가 나오는 것 같다.



[그림 7]

Version 2에서 hash 함수에 개수를 1개에서 10개까지 증가시켜가며, 그에 따라 group의 수도 각각 1개에서 hash 함수 개수까지 증가시켰을 때의 RMSE 값을 구해보았다. Hash 함수가 10개, group의 개수가 8개(따라서 hash 함수를 2,2,1,1,1,1,1,1개로 나누어 각각 중앙값을 구해 평균을 구한다)일 때가 가장 낮은 오차값이 나왔다.