[과제] Bloom Filter & Flajolet-Martin

소프트웨어학부

20203066 박민희

Bloom Filter

import math  
import mmh3  
import random  
  
  
class BloomFilter:  
 def \_\_init\_\_(self,capacity,fp\_prob): #capacity = m,fp\_brob = false positive 비율  
 self.capacity = capacity #언제쓸지 모르니까 인스턴스에 멤버변수로 넣어주기  
 self.fp\_prob = fp\_prob  
 self.bitarray = 0  
 self.n\_bits = math.ceil(-math.log(fp\_prob,math.e)\*capacity/(math.log(2,math.e)\*\*2)) #bitarray 크기인듯?  
 self.n\_hashs = int(self.n\_bits / capacity\* math.log(2,math.e)) #hash 개수  
 #print(self.n\_bits)  
 #print(self.n\_hashs)  
 self.seeds = [random.randint(0,999999) for i in range(self.n\_hashs)]  
  
 def put(self,item):  
 for i in range(self.n\_hashs): #hash 개수 만큼 해시를 돌림  
 pos = mmh3.hash(item,self.seeds[i]) % self.n\_bits  
 self.bitarray |= (1<<pos) #bitarray에다가 set , 1에서 pos만큼 shift시켜서 #그럼 해당위치에 set됩니당.  
  
 def test(self,item): #item이 들어왔을때 그 item을 hash를 해서 pos를 구하고 해당 pos가 전부 다 1이면 있을수도 있다~  
 for i in range(self.n\_hashs):   
 pos = mmh3.hash(item,self.seeds[i]) % self.n\_bits  
   
 if self.bitarray & (1<<pos) == 0: #이게 true면 0, false면 1  
 return False  
 return True  
  
bloom = BloomFilter(10, 0.1)  
  
bloom.put('a')  
bloom.put('b')  
bloom.put('c')  
bloom.put('d')  
bloom.put('e')  
  
print('a',bloom.test('a'))  
print('b',bloom.test('b'))  
print('c',bloom.test('c'))  
print('d',bloom.test('d'))  
print('e',bloom.test('e'))  
print('f',bloom.test('f'))  
print('g',bloom.test('g'))  
print('h',bloom.test('h'))  
print('i',bloom.test('i'))  
print('j',bloom.test('j'))  
print('k',bloom.test('k'))  
print('l',bloom.test('l'))  
print('m',bloom.test('m'))  
print('n',bloom.test('n'))

위 코드의 결과는

a True

b True

c True

d True

e True

f False

g False

h False

i False

j False

k False

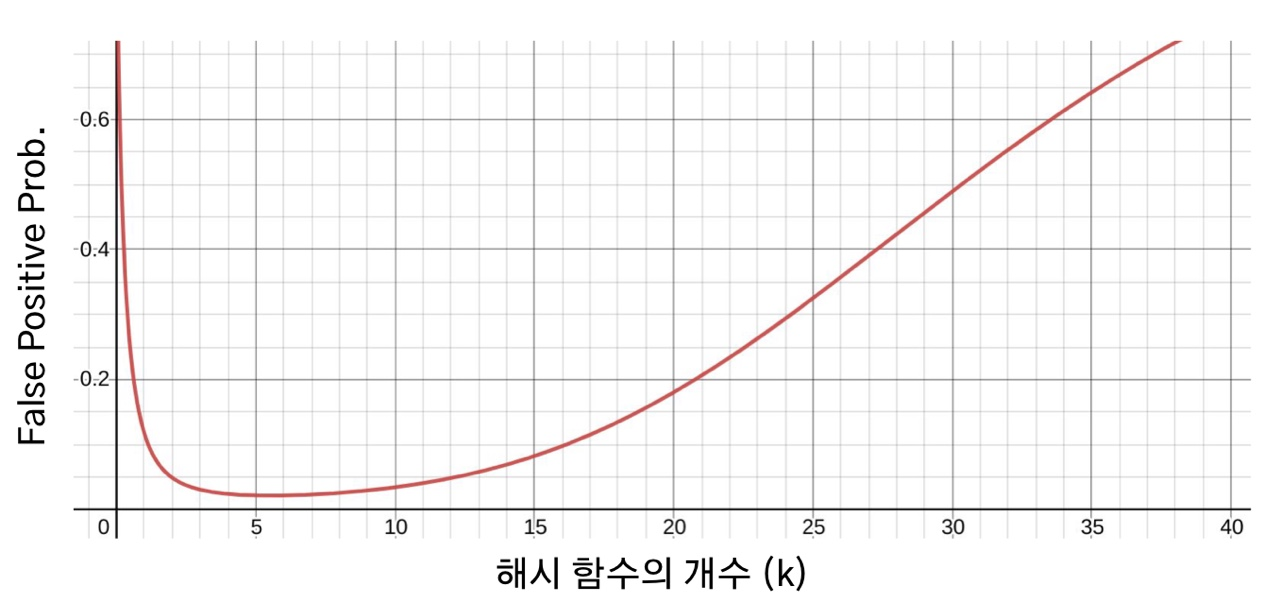
l True

m False

n False

로 출력되었다.

1억명의 사용자 계정이 시스템에 저장되어 있고, 사용자가 회원가입 중에 동일한 계정명이 서버에 존재하는지 즉각확인하는 시스템을 개발할 때 Bloom Filter를 활용한다면 동일한 계정명이 서버에 존재하는지를 잘 확인할수 있을것이다. 동일한 계정명이 서버에 없는데 있다고 나올순 있지만 없는데 있다고 나올수는 없기 때문이다.



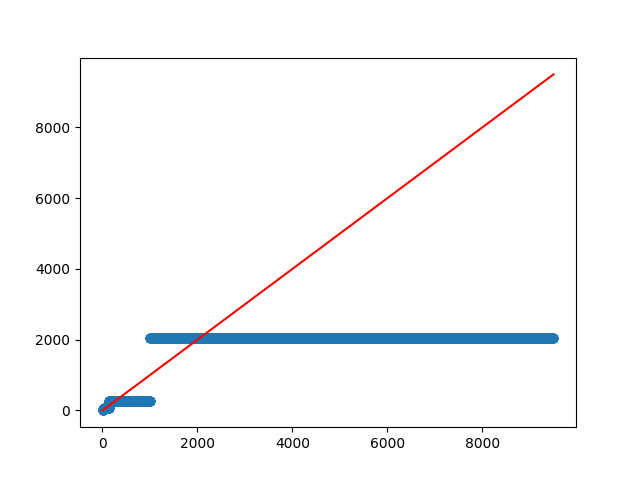
위의 그래프에서 볼수 있듯 False Positive 확률은 해시함수에 따라 다른데 최 저점일때의 해시함수는 n/m ln(2)이다 이때 n은 비트배열의 크기,m은 집합 s의 크기이다.

이로 볼 때 알수 있는 것은 비트배열의 크기가 너무 커져서는 안된다는 것이다. k의 값이 적절할때 False Positive의 값이 최저이다. 그래프에서 k의 값이 5때를 최저라고 생각했을 때 적절한 비트배열의 크기는 n/1억\*ln(2)=5를 풀었을 때 나오는 n의 값이 적절할수 있을거라고 생각한다.

또한 학교 컴퓨터를 사용한사람 중 최근 일주일동안 ecampus에 로그인한 사람만의 이력을 뽑는다고 하였을 때 국민대학교 ecampus사용자 데이터를 이용하여 bloom filter를 만들고 국민대ecampus사용자가 아닌 사람을 첫번째로 확실하게 거를수 있을으므로 bloom filter가 도음이 될수 있을것같다.

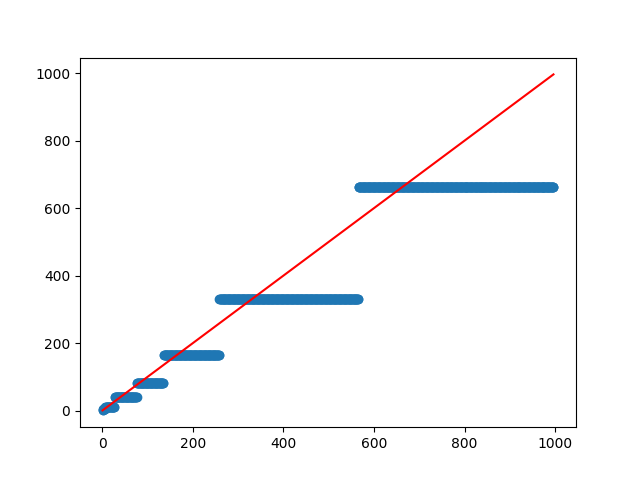
Flajolet-Martin

ver1.

import mmh3  
import math  
import random  
import matplotlib.pyplot as plt  
from tqdm import tqdm  
  
class FM:  
 def \_\_init\_\_(self,domain\_size):  
 self.bitarray = 0  
 self.domain\_size = domain\_size  
 self.n\_bits = math.ceil(math.log2(domain\_size)) #몇개의 bit를 쓸건지?  
 self.mask = (1 << self.n\_bits) -1 #11111111  
 self.seed = random.randint(0,999999)  
  
 def put(self,item): #item들어오면 hash하고 위치찾고 bitarray에서 해당위치 1인애 설정하면 됑  
 h = mmh3.hash(item,self.seed) & self.mask #hash 하는 부분??  
 r = 0  
 if h == 0 : return  
 while (h & (1 << r)) == 0 : r += 1 #1위치 찾기  
 self.bitarray |= (1 << r) #bitarray에 r번 위치에다가 1을 셋팅하는ㄴ겨  
  
 def size(self): #2\*\*R  
 R = self.n\_bits-1  
 while self.bitarray & (1 << R) == 0: R -= 1 # bitarray에서 쩰 큰 1이 나오는 곳 찾는거 ,,  
 #print(self.bitarray)  
 return 2 \*\* R  
  
fm = FM(1000000)  
tset = set() #True set  
  
x = []  
y = []  
  
  
for i in tqdm(range(10000)):  
 item = str(random.randint(0, 100000))  
 fm.put(item)  
 tset.add(item)  
  
 x.append(len(tset))  
 y.append(fm.size())  
  
plt.scatter(x,y)  
plt.plot(x,x,color='r')  
plt.show()  
 #print(f"true: {len(tset)}, estimated: {fm.size()}")

ver2.

import mmh3  
import math  
import random  
import matplotlib.pyplot as plt  
from tqdm import tqdm  
  
class FM:  
 def \_\_init\_\_(self,domain\_size,n\_groups):  
 self.domain\_size = domain\_size  
 self.n\_groups = n\_groups  
 self.n\_bits = math.ceil(math.log2(domain\_size)) #몇개의 bit를 쓸건지?  
 self.mask = (1 << self.n\_bits) -1 #11111111  
 self.seed = 100 #해시함수 개수입미당  
 self.seeds = [random.randint(0,999999) for i in range(self.seed)]  
 self.bitarray = [0 for i in range(self.seed)]  
  
 def put(self,item): #item들어오면 hash하고 위치찾고 bitarray에서 해당위치 1인애 설정하면 됑  
 for i in range(self.seed):  
 h = (mmh3.hash(item,self.seeds[i]) & self.mask) #hash 하는 부분??  
 r = 0  
 if h == 0 : return  
 while (h & (1 << r)) == 0 : r += 1  
 self.bitarray[i] |= (1 << r) #bitarray에 r번 위치에다가 1을 셋팅하는ㄴ겨  
  
 def size(self): #2\*\*R/Φ  
 group = [[] for i in range(self.n\_groups)]  
 R = 0  
 for i in range(self.seed):  
 while self.bitarray[i] & (1 << R) != 0: R += 1 #bitarray에서 처음 0이 나오는 곳 찾는거 ,, 그  
 group[i%self.n\_groups].append(2 \*\* R / 0.77351)  
 jungahng = 0  
 for i in group:  
 i.sort()  
 jungahng += i[len(i)//2]  
  
 return jungahng/self.n\_groups  
  
fm = FM(100000,10)  
tset = set() #True set  
  
x = []  
y = []  
  
  
for i in tqdm(range(100000)):  
 item = str(random.randint(0, 100000))  
 fm.put(item)  
 tset.add(item)  
  
  
 x.append(len(tset))  
 y.append(fm.size())  
  
plt.scatter(x,y)  
plt.plot(x,x,color='r')  
plt.show()  
 #print(f"true: {len(tset)}, estimated: {fm.size()}")

ver1에 비해서 ver2가 확실히 정확도가 높아진 것을 볼 수 있다.

해시 함수의 수를 일정하게 두고 보았을 때 그룹의 수가 많은 것보다 적을 때 더 결과가 조금더 정확히 추측된것 같다. 또한 그룹의 수를 일정하게 두고 해시함수의 수를 변화시켜보았을 때 해시함수가 과하게 많은 것은 의미가 없는 것 같다고 생각되었다. 해시함수가 일정 수를 지나고 난 후에는 결과가 크게 다르지 않다고 느꼈다.