Computer Science 717 S1 (2022)

Assignment 4

Due date May 15, 2022 23:59pm

Answer all of the following questions. There are 10 points, which contribute 10% of your total course marks. Submit a pdf file and a Python script of your answers to Canvas. There is NO automarker for the Python program.

Email spam filtering

You are implementing a new email spam filtering for the University of Auckland. Given that you know the set A of **1 billion** trusted email addresses, e.g., ninh.pham@auckland.ac.nz. If the email comes from one of these trusted addresses, it is not a spam. Otherwise, it will be a spam. Since the server memory is very limited, you cannot keep the list of trusted addresses on its memory (if an email address requires 25 bytes on average, it would take 25 GB to store A in the memory).

After searching on Google, you have found a very memory-efficient solution. That is, instead of keeping A on memory, you will construct a bit array B of size n representing for the set A. You choose n = 8000,000,000 bits so that you need only 1 GB of memory. The construction is as follows.

• B is initialized by 0s.

• You choose a hash function h: a → [0, n). In other words, the input of h will be a string a (e.g., email address) and the hash value will be an integer ranging between 0 and n.

• For each trusted email address a ∈ A, you hash a into one of n buckets, and set that bit to 1. In order words, you simply set B[h(a)] = 1.

The filtering mechanism works as follows.

• When you receive a new email from the address a′, you compute the hash value h(a′).

• If B[h(a′)] = 1, you consider that this email is not a spam and let it go through.

• If B[h(a′)] = 0, you consider that this email is a spam and discard it.

Theoretical questions for measuring the performance of the filtering (5 pts):

1. Illustrate that if a new email from the address a ∈ A, it always gets through (1 pts).

Answer:

When we design the filter, we design a hash function that hashes each email address in set A to an integer h(a), and then sets the $h(a)^{th}$ bit in the bit array to 1.

Ideally (assuming no collision), each email address corresponds to a specific integer. If a new email address a ∈ A, after the same hash function operation, the obtained integer h(a)' must be the

same as the h(a) of a certain email address in set A. That is to say, the h(a)th bit in the bit array must also be 1, i.e., B[h(a)] = 1. So, for a new email address a ∈ A, it can always get through.

2. Given any position 0 ≤ i < n, what is the probability that B[i] = 1 (2 pts).

Answer:

A = 1 billion email addresses.

B = 8 billion bits array.

Ideally (assuming only one hash function and no collision), each email address is hashed to a specific integer, so there will be 1 billion bits in the bit array set to 1, and the probability of B[i] = 1 is 1/8.

However, the reality is that the hash function will not be absolutely perfect, collisions are inevitable, that is, there may be more than one email address hashed to the same integer, so the same position in the bit array is set to 1.

So in general, the probability of B[i] = 1 will be less than 1/8.

Also, if we compute email addresses with multiple different hash functions at the same time, we'll get multiple integers. For example, an email address operated with 3 different hash functions will result in 3 integers, so ideally (assuming no collision), in the bit array, there will be a total of 3 billion bits set to 1, the probability of B[i] = 1 is 3/8. Due to the existence of collision, the actual probability will be less than 3/8.

If we keep increasing the number of hash functions, in extreme cases, the bit array may be all set to 1.

3. Given a spam email from the address a′ ∉ A, what is the probability that it gets through (2 pts).

Answer:

A = 1 billion email addresses.

B = 8 billion bits array.

Ideally (assuming only one hash function and no collision), the hash value of the spam address is different from the hash value of the trust email address, and the hash value of the spam address corresponds to the position in the bit array is 0.

However, it is also possible that the hash value of the spam address is the same as the hash value of the trusted email address.

So, when B[h(a)] = 1, we can't say that the email address **must** be in set A, but that it is **likely** to be in set A. For a spam email from the address a′ ∉ A, the probability that it gets through is 1/8 (often less than 1/8 due to collisions).

There is another situation:

Index of bit array	0	1	2	3	4	5	6	7	8	9
B[h(a)]	0	1	1	1	1	1	1	1	1	1
A = {X, Y, Z}		Z	X	Y	Z	Y	X	Y	X	Z
W			W			W				W

X: h1(x) = 2, h2(x) = 6, h3(x) = 8

Y: h1(y) = 3, h2(y) = 5, h3(y) = 7

Z: h1(z) = 1, h2(z) = 4, h3(z) = 9

W: h1(w) = 2, h2(w) = 5, h3(w) = 9

When we use multiple hash functions, a trusted email address will be hashed into multiple integers. At this time, although W does not belong to set A, the index position obtained by multiple hash operations has been set to 1 by the elements in the set A, so, W will get through by mistake.

We analyze the above problem with a balls and bins model. Assuming we have m balls and n bins, the probability that a bin does not have any ball is $(1-1/n)^m$, so we can infer that the probability that a bin has at least one ball is $1-(1-1/n)^m$.

Since $1-(1-1/n)^m \approx 1-e^{-m/n}$, when A = 1 billion email addresses, B = 8 billion bits array, the probability that $a' \notin A$ gets through (i.e., false positive) is 0.1175.

When we use multiple hash functions, the balls and bins model can be transformed into randomly throwing km balls into n bins, so the false positive is $(1 - e^{-km/n})^k$. If k = 1, the probability that $a' \notin A$ gets through is $(1 - e^{-1/8})^1 = 0.1175$; if k = 2, the probability that $a' \notin A$ gets through is $(1 - e^{-2/8})^2 = 0.048929$.

Practical implementation for measuring the performance of the filtering (5 pts):

Write a Python script to implement this email spam filtering technique given the scaled setting. Your data structure B has size n = 8000,000 bits. For simplicity, assume that your trusted email addresses and spam email are presented as integers. In particular, trusted email addresses A = {1, 2, ..., 1000000} and spam email addresses are any integer x > 1000,000. You are free to choose your hash function to hash an integer into the range [0, n).

1. Verify that any email from the address 1 ≤ a ≤ 1000,000 it always gets through (1 pts).

Answer:

1.1 Create a list that 1 ≤ a ≤ 1000,000.

```
# Email from the address 1 ≤ a ≤ 1000000
white_List = list(range(1, 1000001))
```

1.2 Load to test set.

```
test_Set = white_List
```

1.3 If a ∈ A, the number of trust + 1; otherwise spam + 1.

```
if flag is True:
    # Count the number of passes
    trust = trust + 1
else:
    # Count the number of spams
    spam = spam + 1
```

1.4 The output shows that all addresses passed.

```
Number of pass is: 1000000
Number of NOT pass is: 0
```

For the detail of the Python script, please refer to "COMPSCI 717 A4.ipynb".

2. Compute the probability that a spam email going through your filter given this setting (1 pts).

Answer:

$k = n\ln(2)/m = 8\ln(2) \approx 6$

The false positive probability is: $(1 - e^{-km/n})^k$

Theoretical error rate at k = 6: $(1 - e^{-3/4})^6 \approx 0.021577 \approx 2.16\%$

3. Generate 1000 random integers x > 1000,000 as spam email addresses and compute the number of spam emails going through your filter. Verify this value with your theoretical value from the step 2 (3 pts).

Answer:

1.1 Create a list that 1000 random integers x > 1000,000. To prevent memory overflow warnings, we set the maximum value to 2**32.

```
# 1000 spam email from the address x > 1000000
black_List = np.random.randint(low = 1000001, high = 2**32, size = 1000, dtype = 'i8')
```

1.2 Load to test set.

```
test_Set = black_List
```

1.3 If a spam email goes through the filter, the number of trust + 1; otherwise spam + 1.

```
if flag is True:
    # Count the number of passes
    trust = trust + 1
else:
    # Count the number of spams
    spam = spam + 1
```

1.4 The output shows that some spam email addresses passed by mistake.

```
Number of pass is: 18
Number of NOT pass is: 982
```

1.5 We count the results of 10 tests.

	1	2	3	4	5	6	7	8	9	10
Pass	29	21	25	24	31	29	22	24	25	18
Not Pass	971	979	975	976	969	971	978	976	975	982
False positive	2.9%	2.1%	2.5%	2.4%	3.1%	2.9%	2.2%	2.4%	2.5%	1.8%

The average false positive rate is 2.48%, which is close to the theoretical value of 2.16%. The inevitable collision of hash functions, the program random seed, the randomly generated hash function, and the randomly generated test set will all affect the results.

For the detail of the Python script, please refer to “COMPSCI 717 A4.ipynb”.