Assignment 5

CS-UH-2218: Algorithmic Foundations of Data Science

Assignments are to be submitted in groups of two or three. *Upload the solutions on NYU classes as one PDF file for theoretical assignments and separate source code files for each programming assignment. Submit only one copy per group. Clearly mention the participant names on each file you submit.*

Problem 1 (10 points)**.**
Suppose that we want to use Count-Min Sketch to select heavy hitters in a stream. All items that are $\frac{1}{k}$-heavy hitters should be selected. In addition, any item that is not a $\frac{1}{2k}$-heavy hitter should have only 0.1% chance of being selected.

- Explain how this can be done if the length of the stream m is known in advance.
- How would you modify your algorithm if m is not known in advance?

Remark: Note that Count-Min Sketch itself does not store any elements. It only maintains the counts. The goal here is to identify and store all the heavy hitters.

Problem 2 (10 points)**.**
In the Count-Min Sketch algorithm we discussed in class, when we encounter an element x, we increment all counters associated with that element. Suppose that we only increment the subset of these counters that currently have the minimum value among the counters associated with x. Does this still work? Is this a good idea?

Problem 3 (20 points)**.**
The goal of this exercise is to test the efficacy of the Flajolet-Martin (FM) algorithm and its variants in estimating the number of distinct elements. We will count distinct words and shingles in the text file `norvig.com/big.txt`. A file can be thought of as a stream of elements (words or shingles). We will use the Flajolet-Martin(FM) algorithm to get several estimates of the number of distinct elements in the stream and then look at various ways of combining these estimates.

The following function to converts each word in a file into an integer by concatenating the ASCII representation of its characters and returns the resulting stream.

```
import string
def wordStream(fileName):
  with open(fileName, "r") as infile:
    for line in infile:
      for w in line.strip().lower().split():
        z = 0
        for c in w.strip(string.punctuation):
          z = (z<<8) | ord(c)
        yield z
```

We can compute the number of distinct integers in the stream exactly as follows:

```
def countDistinct(stream):
  M = {}
  for x in stream: M[x]=1
  return len(M.keys())
```

The above code of course uses an amount of memory proportional to the size of the set of distinct words and would not work if this was too large. We will only use it to measure the quality of the estimates we get.

The FM algorithm, as discussed in class, computes a hash value for each integer in the stream and uses the maximum number of trailing zeros seen in hash values to estimate the number of distinct integers. As mentioned in class, it suffices that the hash functions are from a 2-universal family. We will use a 2-universal family $\mathcal{H}$ constructed as follows. Let p be a prime number and let $m \leq p$ be a positive integer. For any $a \in \{1, \cdots, p-1\}$ and $b \in \{0, \cdots, p-1\}$, we define $h_{a,b}(x) = ((ax+b) \bmod p) \bmod m$. $\mathcal{H}$ is the collection of all such functions. To pick a random function from this family, we simply pick a and b randomly from their respective ranges and use $h_{a,b}$.

We will use $p = 9576890767$ and $m = 2^{32}$. In order to get several estimates, we will generate several pairs of (a_i, b_i) where $a_i \in \{1, \cdots, p-1\}$ and $b_i \in \{0, \cdots, p-1\}$ and define $h_i(x) = ((a_i x + b_i) \bmod p) \bmod m$. For each h_i, let $z_i = \max\{zeros(h_i(x)) : x \in S\}$ where S is the set of elements in the stream. As in the lectures, $zeros(y)$ denotes the number of trailing zeros in the binary representation of y. For technical convenience, we will define $zeros(0) = 0$. Write a function `FM_estimates` which takes as input the stream and a number r denoting the required number of estimates and returns an array of independent estimates $[z_0, z_1, \cdots, z_r]$. Your function should look something like:

```
def FM_estimates(stream, r):  # r is the number of estimates needed
  p = 9576890767 # some large enough prime number
  m = 2**32

  # Generate a_i's and b_i's for r random hash functions
  a = [random.randint(1, p) for i in range(r)]
  b = [random.randint(0, p) for i in range(r)]
  z = [0 for i in range(r)] # counts max no. of zeros for each hash fn.

  for x in stream:
    for i in range(r):
      y = ((a[i]*(x % p) + b[i]) % p) % m # random hash function
      # update z[i] if y has more than z[i] trailing 0's

      # YOUR CODE BELOW
      # ....

  return z
```

Feel free to modify the code as necessary. For efficiency reasons, avoid function calls in the loop.

Combining Estimates. We can obtain a set of 100 estimates of the number of distinct words in the file `big.txt` as follows.

```
z = FM_estimates(wordStream("big.txt"),100)
```

There are several ways of combining the estimates to obtain a final estimate. Note that each z_i represents an estimate of 2^{z_i} by the FM algorithm. Try the following ways of combining the estimates and state which seems to be the best:

1. mean of the estimates: $(2^{z_0} + \cdots + 2^{z_{99}})/100$.
2. median of the estimates: median$\{2^{z_0}, \cdots, 2^{z_{99}}\}$.
3. harmonic mean of the estimates: $100/(2^{-z_0} + \cdots + 2^{-z_{99}})$.

Stochastic Averaging. The above approach has the drawback that we need to compute 100 hash functions on each stream element. One way around this is the following. Compute just one hash value for each element and use its first few bits (say r bits) to partition the elements into 2^r groups. Each distinct element is thus assigned a unique group. In each group, we use the remaining bits of the hash value to estimate the number of distinct elements using the FM Algorithm. Thus, we get $K = 2^r$ estimates, each estimate a power of 2. A popular algorithm called the HyperLogLog algorithm then combines these estimates using the harmonic mean of the estimates i.e., if the individual group estimates are $2^{z_0}, \cdots, 2^{z_{K-1}}$, then it returns $K^2/(2^{-z_0} + \cdots + 2^{-z_{K-1}})$. [1]

We pick a function h uniformly at random from the 2-universal family $\mathcal{H}$ constructed before and use it as our hash function. Note that each hash value is a 32 bit number (since $m = 2^{32}$). Of these, we will use the first 6 bits as the group number and remaining 26 for the FM algorithm. Thus, we will have $2^6 = 64$ groups.

Use the HyperLogLog algorithm to estimate the number of distinct 9-shingles[2] in the file `big.txt`. How good is the estimate?

[1] The algorithm also does a few corrections but will ignore those here. See `http://algo.inria.fr/flajolet/Publications/FlFuGaMe07.pdf`.

[2] Recall that a k-shingle is any substring of the text with k consecutive characters (including space).