**Assignment 4**

In this assignment, we will explore countmin sketches and bloom filters. We will use two text files great-gatsby-fitzgerald.txt and war-and-peace-tolstoy.txt to load up the text of two famous novels courtesy of Project Guttenberg.

We will explore two tasks:

* Counting the frequency of words of length 5 or more in both novels using a count-min sketch
* Using a bloom filter to approximately count how many words in the War and Peace novel already appears in the Great Gatsby.

**Step 1: Making a Universal Hash Family (Already Done For You)**

We will use a family of hash function that first starts by (a) generating a random prime number 𝑝 (we will use the Miller-Rabin primality test for this purpopse); (b) generating random numbers a, b between 2 and p-1.

The hash function ℎ𝑎,𝑏,𝑝(𝑛)=(𝑎𝑛+𝑏)mod𝑝.

Note that this function will be between 0 and p-1. We will need to also make sure to take the hash value modulo 𝑚 where 𝑚 is the size of the hashtable.

To hash strings, we will first use python's inbuilt hash function and then use ℎ𝑎,𝑏,𝑝 on the result.

As a first step, we will generate a random prime number.

**(A) Generate Random Prime Numbers**

*# Python3 program Miller-Rabin randomized primality test*

*# Copied from geeksforgeeks: https://www.geeksforgeeks.org/primality-test-set-3-miller-rabin/*

**import** random

​

*# Utility function to do*

*# modular exponentiation.*

*# It returns (x^y) % p*

**def** power(x, y, p):

*# Initialize result*

res **=** 1;

*# Update x if it is more than or*

*# equal to p*

x **=** x **%** p;

**while** (y **>** 0):

*# If y is odd, multiply*

*# x with result*

**if** (y **&** 1):

res **=** (res **\*** x) **%** p;

​

*# y must be even now*

y **=** y**>>**1; *# y = y/2*

x **=** (x **\*** x) **%** p;

**return** res;

​

*# This function is called*

*# for all k trials. It returns*

*# false if n is composite and*

*# returns false if n is*

*# probably prime. d is an odd*

*# number such that d\*2<sup>r</sup> = n-1*

*# for some r >= 1*

**def** miillerTest(d, n):

*# Pick a random number in [2..n-2]*

*# Corner cases make sure that n > 4*

a **=** 2 **+** random.randint(1, n **-** 4);

​

*# Compute a^d % n*

x **=** power(a, d, n);

​

**if** (x **==** 1 **or** x **==** n **-** 1):

**return** **True**;

​

*# Keep squaring x while one*

*# of the following doesn't*

*# happen*

*# (i) d does not reach n-1*

*# (ii) (x^2) % n is not 1*

*# (iii) (x^2) % n is not n-1*

**while** (d **!=** n **-** 1):

x **=** (x **\*** x) **%** n;

d **\*=** 2;

​

**if** (x **==** 1):

**return** **False**;

**if** (x **==** n **-** 1):

**return** **True**;

​

*# Return composite*

**return** **False**;

​

*# It returns false if n is*

*# composite and returns true if n*

*# is probably prime. k is an*

*# input parameter that determines*

*# accuracy level. Higher value of*

*# k indicates more accuracy.*

**def** isPrime( n, k):

*# Corner cases*

**if** (n **<=** 1 **or** n **==** 4):

**return** **False**;

**if** (n **<=** 3):

**return** **True**;

​

*# Find r such that n =*

*# 2^d \* r + 1 for some r >= 1*

d **=** n **-** 1;

**while** (d **%** 2 **==** 0):

d **//=** 2;

​

*# Iterate given nber of 'k' times*

**for** i **in** range(k):

**if** (miillerTest(d, n) **==** **False**):

**return** **False**;

​

**return** **True**;

​

*# Driver Code*

*# Number of iterations*

k **=** 4;

​

print("All primes smaller than 100: ");

**for** n **in** range(1,100):

**if** (isPrime(n, k)):

print(n , end**=**" ");

​

*# This code is contributed by mits (see citation above)*

​

**Step 2: Universal Hash Families**

We will provide three useful functions for you:

* get\_random\_hash\_function: Generate triple of numbers (p, a, b) at random, where p is prime, a and b are numbers between 2 and p-1. The hash function ℎ𝑝,𝑎,𝑏(𝑛) is given by (𝑎𝑛+𝑏)mod𝑝.
* hashfun: apply the random hash function on a number num.
* hash\_string: apply the hash function on a string hstr. Note that the result is between 0 and p-1. If your hash table has size m, you should take a mod m on this result where you call hash\_string.

Please use these functions in your code below.

*# Get a random triple (p, a, b) where p is prime and a,b are numbers betweeen 2 and p-1*

**def** get\_random\_hash\_function():

n **=** random.getrandbits(64)

**if** n **<** 0:

n **=** **-**n

**if** n **%** 2 **==** 0:

n **=** n **+** 1

**while** **not** isPrime(n, 20):

n **=** n **+** 1

a **=** random.randint(2, n**-**1)

b **=** random.randint(2, n**-**1)

**return** (n, a, b)

​

*# hash function fora number*

**def** hashfun(hfun\_rep, num):

(p, a, b) **=** hfun\_rep

**return** (a **\*** num **+** b) **%** p

​

*# hash function for a string.*

**def** hash\_string(hfun\_rep, hstr):

n **=** hash(hstr)

**return** hashfun(hfun\_rep, n)

**Step 3: Loading Data**

We are going to load two files great-gatsby-fitzgerald.txt and war-and-peace-tolstoy.txt to load up the text of two famous novels courtesy of Project Guttenberg. We will filter all wordsd of length >= 5 and also count the frequency of each word in a dictionary. This will be fast because it is going to use highly optimized hashtable (dictionaries) built into python.

*# Let us load the "Great Gatsby" novel and extract all words of length 5 or more*

filename **=** 'great-gatsby-fitzgerald.txt'

file **=** open (filename,'r')

txt **=** file.read()

txt **=** txt.replace('\n',' ')

words**=** txt.split(' ')

longer\_words\_gg **=** list(filter(**lambda** s: len(s) **>=** 5, words))

print(len(longer\_words\_gg))

*# Let us count the precise word frequencies*

word\_freq\_gg **=** {}

**for** elt **in** longer\_words\_gg:

**if** elt **in** word\_freq\_gg:

word\_freq\_gg[elt] **+=** 1

**else**:

word\_freq\_gg[elt] **=** 1

print(len(word\_freq\_gg))

*# Let us load the "War and Peace" novel by Tolstoy translation and extract all words of length 5 or more*

filename **=** 'war-and-peace-tolstoy.txt'

file **=** open (filename,'r')

txt **=** file.read()

txt **=** txt.replace('\n',' ')

words**=** txt.split(' ')

longer\_words\_wp **=** list(filter(**lambda** s: len(s) **>=** 5, words))

print(len(longer\_words\_wp))

word\_freq\_wp **=** {}

**for** elt **in** longer\_words\_wp:

**if** elt **in** word\_freq\_wp:

word\_freq\_wp[elt] **+=** 1

**else**:

word\_freq\_wp[elt] **=** 1

print(len(word\_freq\_wp))

**Problem 1: Implement count-min sketch**

Implement CountMinSketch class below where num\_counters is the number of counters. You are given the constructor that already generates a random representative of a hash function family. Implement the functions:

* increment
* approximateCount.

Please read the constructor carefully: it initializes the counters and generates the hash function for you. Also, when you call hash\_string function defined previously, do not forget to take result modulo m.

*# Class for implementing a count min sketch "single bank" of counters*

**from** \_\_future\_\_ **import** annotations

**from** typing **import** List

​

**class** CountMinSketch:

*# Initialize with `num\_counters`*

**def** \_\_init\_\_ (self, num\_counters):

self.m **=** num\_counters

self.hash\_fun\_rep **=** get\_random\_hash\_function()

self.counters **=** [0]**\***self.m

**def** get\_hash(self, word: int) **->** int:

raw\_hash: int **=** hash\_string(self.hash\_fun\_rep, word)

**return** raw\_hash **%** self.m

*# function: increment*

*# given a word, increment its count in the countmin sketch*

**def** increment(self, word: str) **->** **None**:

word\_hash: int **=** self.get\_hash(word)

self.counters[word\_hash] **+=** 1

*# function: approximateCount*

*# Given a word, get its approximate count*

**def** approximateCount(self, word) **->** int:

word\_hash: int **=** self.get\_hash(word)

**return** self.counters[word\_hash]

*# We will now implement the algorithm for a bank of k counters*

​

*# Initialize k different counters*

**def** initialize\_k\_counters(k, m):

**return** [CountMinSketch(m) **for** i **in** range(k)]

​

*# Function increment\_counters*

*# increment each of the individual counters with the word*

**def** increment\_counters(count\_min\_sketches, word):

**for** cms **in** count\_min\_sketches:

cms.increment(word)

**return** count\_min\_sketches

*# Function: approximate\_count*

*# Get the approximate count by querying each counter bank and taking the minimum*

**def** approximate\_count(count\_min\_sketches, word):

**return** min([cms.approximateCount(word) **for** cms **in** count\_min\_sketches])

​

**%**matplotlib inline

**from** matplotlib **import** pyplot **as** plt

​

*# Let's see how well your solution performs for the Great Gatsby words*

cms\_list **=** initialize\_k\_counters(5, 1000)

**for** word **in** longer\_words\_gg:

increment\_counters(cms\_list, word)

​

discrepencies **=** []

**for** word **in** longer\_words\_gg:

l **=** approximate\_count(cms\_list, word)

r **=** word\_freq\_gg[word]

**assert** ( l **>=** r)

discrepencies.append( l**-**r )

plt.hist(discrepencies)

​

**assert**(max(discrepencies) **<=** 200), 'The largest discrepency must be definitely less than 200 with high probability. Please check your implementation'

print('Passed all tests: 10 points')

*# Let's see how well your solution performs for the War and Peace*

cms\_list **=** initialize\_k\_counters(5, 5000)

**for** word **in** longer\_words\_wp:

increment\_counters(cms\_list, word)

​

discrepencies **=** []

**for** word **in** longer\_words\_wp:

l **=** approximate\_count(cms\_list, word)

r **=** word\_freq\_wp[word]

**assert** ( l **>=** r)

discrepencies.append( l**-**r )

​

plt.hist(discrepencies)

print('Passed all tests: 5 points')

**Problem 1B**

Check the data obtained above with calculations along the lines of what was done in class. If we had 5 banks of counters with 5000 counters each and a uniform hash function family, what is the probability that when counting a total of 𝑁=2.5×105 words, we have a discrepency by 80 or more.

This problem will not be graded but simply for you to understand the calculations involved.

YOUR ANSWER HERE The probability of having a discrepancy of 80 or more in any of the 5 banks is approximately 0.4207 or 42.07%.

**Problem 2: Using a Bloom Filter to Count Common Words.**

In this problem, we will implement a Bloom filter to count how many elements of longer\_words\_wp (the words of length 5 or more in War and Peace) appear in the Great-Gatsby novel. To do so, we will do the following:

* Instantiate a Bloom filter with number of bits n and number of hash functions k.
* Insert all words from great-gatsby into the filter.
* For each word from war and peace, check membership in the Bloom filter and count the number of yes answers.

**from** \_\_future\_\_ **import** annotations

**from** typing **import** List

​

**class** BloomFilter:

**def** \_\_init\_\_(self, nbits, nhash):

self.bits: List[bool] **=** [**False**]**\***nbits *# Initialize all bits to fals*

self.m: int **=** nbits

self.k: int **=** nhash

*# get k randdom hash functions*

self.hash\_fun\_reps: List[(int, int, int)] **=** \

[get\_random\_hash\_function() **for** i **in** range(self.k)]

**def** get\_hash(self, hash\_fun\_rep: (int, int, int), word: int) **->** int:

raw\_hash: int **=** hash\_string(hash\_fun\_rep, word)

**return** raw\_hash **%** self.m

*# Function to insert a word in a Bloom filter.*

**def** insert(self, word) **->** **None**:

**for** \_k **in** range(self.k):

word\_hash **=** self.get\_hash(self.hash\_fun\_reps[\_k], word)

self.bits[word\_hash] **=** **True**

*# Check if a word belongs to the Bloom Filter*

**def** member(self, word) **->** bool:

word\_hashes: List[int] **=** [self.get\_hash(self.hash\_fun\_reps[\_k], word) **for** \_k **in** range(self.k)]

**return** all([self.bits[num\_bit] **for** num\_bit **in** word\_hashes])

​

*#do the exact count*

*# it is a measure of how optimized python data structures are under the hood that*

*# this operation finishes very quickly.*

all\_words\_gg **=** set(longer\_words\_gg)

exact\_common\_wc **=** 0

**for** word **in** longer\_words\_wp:

**if** word **in** all\_words\_gg:

exact\_common\_wc **=** exact\_common\_wc **+** 1

print(f'Exact common word count = {exact\_common\_wc}')

*# Try to use the same using a bloom filter.*

bf **=** BloomFilter(100000, 5)

**for** word **in** longer\_words\_gg:

bf.insert(word)

**for** word **in** longer\_words\_gg:

**assert** (bf.member(word)), f'Word: {word} should be a member'

​

common\_word\_count **=** 0

**for** word **in** longer\_words\_wp:

**if** bf.member(word):

common\_word\_count**=** common\_word\_count **+** 1

print(f'Number of common words of length >= 5 equals : {common\_word\_count}')

**assert** ( common\_word\_count **>=** exact\_common\_wc)

print('All Tests Passed: 10 points')

**Problem 2 B**

Given a Bloom filter with 𝑚=100000 bits and 𝑘=5 hash functions that map each key uniformly at random to one of the bits (assumption), estimate the probability that 𝑘 bits 𝑖1,…,𝑖𝑘 are simultaneously set when 𝑛=10000 words are inserted. Assume that whether or not a particular bit is set is independent of another.

YOUR ANSWER HERE The probability that the k=5k=5 specific bits are simultaneously set after n= 10000n=10000 words are inserted into the Bloom filter is approximately 0.0102 or 1.02%.

**Manually Graded Solutions**

**Problem 1 B**

Note that for each word we have 𝔼(approxCount(𝑤𝑜𝑟𝑑)−count(𝑤𝑜𝑟𝑑))≤𝑁𝑚. The probability that for some word, the approximate count differs from the real one by at least 80 for one of the counter banks is bounded by Markov Inequality as:

ℙ(approxCount(𝑤𝑜𝑟𝑑)−count(𝑤𝑜𝑟𝑑)≥80)≤𝔼(approxCount(𝑤𝑜𝑟𝑑)−count(𝑤𝑜𝑟𝑑))80≤2.5×10580×5000≈58

.

The probability that this happens for all five counter banks is bounded by (58)5≈0.095.

However, this bound happens to be not so tight. Empirically, we see that this happens for roughly one word out of the nearly quarter million words in the corpus.

**Problem 2 B**

The probability that any given bit is not set when 𝑛 words are inserted is (1−1𝑚)𝑘𝑛=(1−1100000)5×10000=𝑒−0.5.

The probability that all five bits are simultaneously set is (1−𝑒−0.5)5=0.009.

Therefore, we will expect the false positive rate to be roughly 1%.

**That's All Folks!**

Thank you for stopping by !! -Sulay Cay ☺