Bloom过滤器

大致思路

python的hashlib标准库提供了MD5、SHA1、SHA256等密码学Hash函数

python的random库提供了产生随机数的方法

random产生的是伪随机数或者说是用一种复杂的方法计算得到的序列值，因此每次运算时需要一个不同的种子值。种子值不同，得到的序列值也不同。而种子值相同，序列值就相同。也就是说，random可以完成hash值到指定区间整数值的映射

确定参数

43631	home.trea	1
43632	medium.cc	1
43633	medium.cc	1
43634	www.linkec	1
43635	medium.cc	1
43636	medium.cc	1
43637	medium.cc	1
43638	emailmark	1
43639	www.mailc	1
43640	www.printl	1
43641		
43642		
43643		
43644		
43645		

查阅csv文件，其中的数据量为43639个（这v不是n）

看样子，csv文件里的1代表无害，-1代表有害

要插入，即有害的，共有n=10328个

根据公式 $$m = -\frac{nlnP(true)}{(ln2)^2}$$

期待的假阳性概率P = 0.01，那么可以算出m =97065

根据公式 $$k = \frac{m}{n}\ln 2$$,算出来k = 6.57。 使用6或者7个Hash函数

> 一开始做错了，取了k=3作为初始值，不过后面有k = 6、7等数的实例

代码

由hash值映射到指定区间整数

```
def func(Length, HashValue):
    random.seed(HashValue)
    return random.randint(0, Length-1)
```

Bloom类

```
class Bloom:
    def __init__(self, length_of_bloom):
        self.vector = []
        self.length = length_of_bloom
        self.hash_algorithm_list = ['md5', 'sha1', 'sha256']
        for i in range(0, length_of_bloom):
            self.vector.append(0)
```

Bloom类拥有一个列表做成的向量vector，一个存放着此过滤器使用了哪些hash算法的列表

输入length后，vector会被初始化为有length个0

Bloom对象的向量初值设置

```
    def set_value_with_certain_hash_algorithm(self, csv_filename, algorithm):
        csvfile = open(csv_filename, mode='r')
        lines = csv.reader(csvfile)

        for line in lines:
            url, isMalicious = line
            if isMalicious == '-1':
                Hash = hashlib.new(name=algorithm, data=url.encode(encoding='utf-
8'))
                HashValue = Hash.hexdigest()
                index = func(Length=self.length, HashValue=HashValue)
                self.vector[index] = 1


    def set_value(self, csv_filename):
```

```
        for algorithm in self.hash_algorithm_list:
            self.set_value_with_certain_hash_algorithm(csv_filename=csv_filename,
algorithm=algorithm)
```

两个都是Bloom类的成员函数

验证

```
    def filter(self, url):#接受一个url，返回0代表通过，1代表拦截
        for name in self.hash_algorithm_list:
            Hash = hashlib.new(name=name, data=url.encode(encoding='utf-8'))
            HashValue = Hash.hexdigest()
            index = func(Length=self.length, HashValue=HashValue)
            if self.vector[index] == 0:
                return 0
        return 1
```

再加一个verify函数做接口，对csv文件里的所有地址都验证一遍

```
    def verify(self,csv_filename):
        harmless_counter = 0#无害的总数
        harmful_counter = 0#有害的总数
        error_in_harmless = 0#无害的却过滤了的数量
        error_in_harmful = 0#有害但没过滤的数量


        csvfile = open(csv_filename, mode='r')
        lines = csv.reader(csvfile)

        for line in lines:
            url, isMalicious = line
            if isMalicious == '1':#无害
                harmless_counter += 1
                if self.filter(url) == 1:
                    error_in_harmless += 1
            elif isMalicious == '-1':#有害
                harmful_counter += 1
                if self.filter(url) == 0:
                    error_in_harmful += 1

        print('无害总数:', harmless_counter, ' 发生错误数:', error_in_harmless, '错误
率:', error_in_harmless / harmless_counter)
        print('有害总数:', harmful_counter, ' 发生错误数:', error_in_harmful, '错误
率:', error_in_harmful / harmful_counter)
```

参数调整及运行结果

控制k不变 调整m

恰好为公式值的情况

取k = 6 m =97065

```
length =  97065
Hash函数个数(k) =  6
无害总数: 33312  发生错误数: 343 错误率: 0.010296589817483189
有害总数: 10327  发生错误数: 0 错误率: 0.0
```

错误率正好是我们期望的错误率0.01

远小于公式值的情况

```
length =  30000
Hash函数个数(k) =  6
无害总数: 33312  发生错误数: 14864 错误率: 0.4462055715658021
有害总数: 10327  发生错误数: 0 错误率: 0.0
```

错误率极高

远大于公式值的情况

```
length =  150000
Hash函数个数(k) =  6
无害总数: 33312  发生错误数: 44 错误率: 0.0013208453410182516
有害总数: 10327  发生错误数: 0 错误率: 0.0

length =  200000
Hash函数个数(k) =  6
无害总数: 33312  发生错误数: 10 错误率: 0.0003001921229586936
有害总数: 10327  发生错误数: 0 错误率: 0.0

length =  4000000
Hash函数个数(k) =  6
无害总数: 33312  发生错误数: 0 错误率: 0.0
```

```
有害总数: 10327  发生错误数: 0 错误率: 0.0
```

当m取足够大，已经一个错误都不发生了

控制m不变，改变k

```
self.hash_algorithm_list = ['md5', 'sha1', 'sha256']

因此，要改变k的值，只需要在这个列表里增删就好了
```

m保持公式值，k=4

```
self.hash_algorithm_list = ['md5', 'sha1', 'sha256', 'sha384']

length =  97065
Hash函数个数(k) =  4
无害总数: 33312  发生错误数: 459 错误率: 0.013778818443804035
有害总数: 10327  发生错误数: 0 错误率: 0.0
```

错误率跟k=6区别不大

m保持公式值，k=5

```
self.hash_algorithm_list = ['md5', 'sha1', 'sha256', 'sha384', 'sha512']

length =  97065
Hash函数个数(k) =  5
无害总数: 33312  发生错误数: 372 错误率: 0.0111671469740634
有害总数: 10327  发生错误数: 0 错误率: 0.0
```

m保持公式值，k=7

```
self.hash_algorithm_list = ['md5', 'sha1', 'sha224', 'sha256', 'sha384',
'sha512', 'blake2b']

length =  97065
Hash函数个数(k) =  7
无害总数: 33312  发生错误数: 360 错误率: 0.010806916426512969
有害总数: 10327  发生错误数: 0 错误率: 0.0
```

m保持公式值，k=9

```
self.hash_algorithm_list = ['md5', 'sha1', 'sha224', 'sha256', 'sha384',
'sha512', 'blake2b', 'blake2s', 'sha3_224']

length =  97065
Hash函数个数(k) =  9
无害总数: 33312  发生错误数: 416 错误率: 0.012487992315081652
有害总数: 10327  发生错误数: 0 错误率: 0.0
```

一些剩余问题

没有考虑哈希函数对结果的影响