Department of Computer Science and Engineering (Data Science)

Machine Learning – IV

Experiment 3

Name: Umang Kirit Lodaya **SAP ID: 60009200032**

Batch: D11

Aim:

To implement and understand the Bloom Filter algorithm, a space-efficient probabilistic data structure used for testing whether an element is a member of a set.

Theory:

Introduction to Bloom Filter Algorithm:

- A Bloom Filter is a space-efficient data structure designed for membership testing in sets.
- It uses multiple hash functions and a bit array to represent elements in the set.

Algorithm Overview:

- **Bit Array:**
  - Use a bit array of size *m* to represent the set, initially setting all bits to 0.
- **Hash Functions:**
  - Utilize *k* independent hash functions that map elements to *m* different positions in the bit array.
- **Insertion:**
  - To insert an element, hash it using the *k* hash functions and set the corresponding bits in the bit array to 1.
- **Membership Test:**
  - To test if an element is in the set, hash it using the *k* hash functions and check if all corresponding bits in the bit array are set to 1.

Step-by-Step Implementation:

- **Step 1: Bit Array Initialization**
  - Implement the initialization of a bit array with all bits set to 0.
- **Step 2: Hash Function Implementation**
  - Create *k* independent hash functions for hashing elements to positions in the bit array.
- **Step 3: Insertion Operation**
  - Implement the insertion operation by setting the bits in the bit array based on the hash values of the element.
- **Step 4: Membership Test Operation**
  - Implement the membership test operation by checking the bits in the bit array based on the hash values of the element.
- **Step 5: Experimentation with Hash Functions**
  - Explore the impact of different hash functions on the false positive rate of the Bloom Filter.

Implementation Tips:

- Choose hash functions that distribute elements evenly to minimize collisions.
- Determine an optimal size for the bit array and number of hash functions based on the expected number of elements and desired false positive rate.

Lab Experiments to be Performed in This Session:

Execute the Bloom's Filter on a dataset to gain insights into its functionality and operation.

NAME: UMANG KIRIT LODAYA

SAP ID: 60009200032

BATCH: K1 / D11

In [1]:

```
class BloomFilter:
    def __init__(self):
        self._hash_functions = input('ENTER HASH FUNCTIONS (COMMA SEPERATED)
        self._hash_functions = [h.strip() for h in self._hash_functions.spli
        self._n = max([int(x.split('%')[1].strip()) for x in self._hash_func
        self._filter = [0 for i in range(self._n)]

    def insert(self, val: int):
        for i in [eval(h, {'x': val}) for h in self._hash_functions]:
            self._filter[i] = 1

    def check(self, val: int) -> bool:
        return min([self._filter[i] for i in [eval(h, {'x': val}) for h in s
```

In [2]:

```
BF = BloomFilter()
```

```
ENTER HASH FUNCTIONS (COMMA SEPERATED): (2 * x) % 3, (x + 3) % 3
```

In [3]:

```
to_insert = [5, 8]

for val in to_insert:
    BF.insert(val)
```

In [4]:

```
FP = 0; TP = 0; TN = 0

to_check = [7, 9]

for val in to_check:
    isIN = BF.check(val)
    print(val, isIN)

    if isIN and val in to_insert:
        TP += 1

    elif not isIN and val not in to_insert:
        TN += 1

    elif isIN and val not in to_insert:
        FP += 1

print()
print(f"TRUE POSITIVE: {TP} / {len(to_check)}")
print(f"TRUE NEGATVE: {TN} / {len(to_check)}")
print(f"FALSE POSITIVE: {FP} / {len(to_check)}")
```

```
7 True
9 False

TRUE POSITIVE: 0 / 2
TRUE NEGATVE: 1 / 2
FALSE POSITIVE: 1 / 2
```