Streaming. 1

Reservoir Sampling.

Para un stream infinito queremos guardar una muestra de tamaño constante "k", si la cantidad de datos vistos hasta el momento es "n" entonces queremos que la probabilidad de que un dato esté en la muestra sea de $\frac{k}{n}$.

Algoritmo:

- Para los primeros "k" elementos todos se almacenan.
- Luego a partir del elemento $k+1$ todos tienen una probabilidad $p = \frac{k}{n}$ de entrar y reemplazar uno de los k datos almacenados.

Podemos utilizar algoritmos usuales sobre estos muestra finita y volver a suponer que es estadísticamente válida

...

Momentos de un stream.

Definimos el momento de orden k como

$$M^k(S) = \sum_i M_i^k$$

Donde M_i es la frecuencia de cada dato i

Momento orden 0:

Es la cantidad de elementos diferentes observados hasta el momento en el Stream.

Es difícil de calcular exactamente pero podemos estimarlo con algunos algoritmos (Flajolet-Martin o Hyperloglog).

Momento orden 1:

Cantidad de datos que hemos observado en el Stream. Fácil de calcular.

Momento orden 2:

Lo llamamos "número sorpresa" de un Stream y es un indicador de si los datos del Stream se distribuyen de forma pareja o si algún dato aparece muchas más veces que otro. Usamos el algoritmo AMS (Alon Matias y Szegedy)

Flajolet - Martin.

Calcula la cantidad de elementos distintos observados hasta el momento en un Stream.

Algoritmo

Streaming 2.

A cada dato se le aplica una función de hashing que genera un número de m bits (m mayor a $\log_2(n)$ siendo n la cantidad de datos observados) y se observa con cuantos bits en 0 comienza el resultado de la función.

En memoria se mantiene la cantidad máxima de bits 0 observados que llamamos "r"

La cantidad de elementos diferentes en el Stream puede estimarse como 2^r.

AMS

Se define un número "k" de estimadores a mantener en memoria. Cada estimador k tiene 2 campos: valor y cantidad.

Por cada elemento observado del Stream si el elemento está en alguno de los variables entonces incrementar esa variable en 1.

El momento de orden 2 se estima como el promedio de $n(2c_i - 1)$ donde n es la cantidad de datos vistos hasta el momento y c_i es la cantidad de cada estimador k.

~~Para mantener los "k" estimadores usaremos el algoritmo de reservoir sampling.~~

Cada estimador estima el modelo de orden 2 del stream mediante:

$$M^2(5) \simeq n\,(2\,c_i - 1)$$

siendo c_i el contador y n la cantidad de elementos del stream. El igual que Flajolet - Martin vamos a agrupar los estimadores en grupos de m estimadores cada uno siendo el resultado final la mediana del promedio de cada grupo.

Para mantener los "k" estimadores usaremos el algoritmo de reservoir sampling.

Bloom Filters.

Dado un stream queremos saber si los elementos que observamos en el mismo pertenecen o no a un cierto conjunto de elementos predefinidos.

Algoritmo

- Contamos con un vector de "m" bits y k funciones de hashing $0 \dots m-1$.
- Para agregar un elemento al filtro

Streaming 3

Le aplicamos las funciones de hashing y luego encendemos en 1 los bits apuntados por las funciones.

- Le pretenden k o menos bits según haya o no colisión.

- Para verificar si un dato pertenece a nuestro conjunto aplicamos las funciones de hashing al elemento y verificamos si todos los bits están en 1:

° Si alguno está en 0 entonces el elemento no pertenece al conjunto.

° Si están todos en 1 el elemento pertenece al conjunto con una cierta probabilidad "p"

$$m \text{ óptimo}: \quad m = -\frac{n \ln(\varepsilon)}{(\ln(2))^2}$$

siendo ε la probabilidad que queremos para un falso positivo.

$$k \text{ óptimo}: \quad k = \frac{m}{n} \ln(2)$$

Se puede estimar la cantidad de elementos insertados al filtro como:

$$E = \frac{-\left(\frac{m}{m}\right) \ln\left(1 - \left(\frac{X}{m}\right)\right)}{k}$$

$\rightarrow X$: cantidad de bits en 1.

Counting Filters

Una modificación de bloom filters que permite eliminar elementos del filtro.

En vez de tener un bitmap, mantenemos en cada posición del filtro un contador. Para agregar un elemento del filtro incrementamos en uno las posiciones indicadas por las funciones de hash. Para eliminar un elemento restamos uno a las posiciones indicadas. Para verificar si un elemento se encuentra en el filtro, verificamos que las posiciones indicadas tengan valores distintos de cero.

Count Min Sketch.

Uso de counting filters para encontrar los elementos más frecuentes en un stream

Se usan "d" filtros en total, cada uno de "w" enteros y asociados a una función de hashing.

Cuando se observa un elemento del stream se le aplican las "d" funciones de hashing. y en cada uno de los "d" counting filters se incrementa la posición indicada por la función de hashing.

Para estimar la frecuencia de un cierto elemento lo que se hace es hashear con

los "d" funciones de hashing, recuperar los valores de los filtros y tomar el mínimo como su valor de frecuencia.

Para llevar registro de los "i" elementos más frecuentes en un stream:

- Reservar memoria para i elementos
- A medida que procesamos el stream actualizamos los filtros y calculamos el count-min del elemento.
- Si el mismo supera al menor de la lista de los i elementos más frecuentes lo reemplazamos.

Cuckoo Filters.

Estructura similar a la de bloom filters pero utilizando cuckoo hashing.

Diferencias:

- Menor espacio
- Más rápidos para consultas.
- Más lentos para inserción.
- Permiten borrar.

No se almacenan los clave sino que sus fingerprint de 6-8 bits. Se almacenan en una tabla de cuckoo hashing de m buckets con hasta b fingerprints por bucket.

Como necesitamos dos funciones de hashing usamos:

$h_1(x) = hash(x)$

$h_2(x) = h_1(x)$ XOR $hash(fingerprint(x))$.

Si M genera un ciclo en la tabla de cuckoo entonces hay que relocalizar elementos. Al relocalizar un ítem existente "i", su otra posición posible "j" en la tabla de cuckoo que estamos gestionando del bucket "i" se calcula como:

$$j = i \text{ XOR } hash(fingerprint(i)).$$

El algoritmo de búsqueda es:

```
f = fingerprint(x)
i1 = hash(x)
i2 = i1 XOR hash(f)
if Table[i1] or Table[i2] has f then:
    return True
return False.
```

Si queremos eliminar realizamos el algoritmo de búsqueda. Si aparece el fingerprint en alguna posición eliminamos 1 copia, cualquiera es suficiente.