

Filtros Bloom

Son una estructura de datos probabilística y optimizada. Se usan para encontrar si un objeto pertenece o no a un dataset. Optimiza este tipo de peticiones usando funciones hash en los elementos a procesar. Cuando el resultado de una petición es positivo, entonces el objeto posiblemente pertenezca al dataset en cuestión, de todas formas pueden ocurrir falsos positivos. Cuando el resultado es negativo, entonces el objeto no pertenece al dataset, no hay falsos negativos. Esta pensado para volúmenes de datos a gran escala.

Un filtro bloom puede ser definido como una tabla o array compuesta por m bits. Inicialmente todos los bits están inicializados a 0. Para añadir un elemento x a la tabla, se usan funciones hash k para encontrar su posición en la tabla y se establecen dichos bits a 1. En un filtro bloom clásico no se pueden eliminar items.

Parametrización de los filtros de bloom

La probabilidad de falsos positivos para un elemento que no pertenece al set es:

$$\epsilon = (1 - (1 - \frac{1}{m})^{nk})^k \approx (1 - e^{-kn/m})^k$$

Por lo tanto, el numero de funciones hash óptimo es:

$$k = \frac{m}{n} \ln 2$$

Y el tamaño del filtro de bloom puede ser determinado como:

$$m = - \frac{n \ln \epsilon}{(\ln 2)^2}$$

n es el número de objetos almacenados dentro del filtro de bloom.

Propiedades de los Filtros de Bloom

Podemos estimar el número de elementos en un filtro de bloom F como:

$$|F| \approx - \frac{m}{l} \ln(1 - \frac{\sum_{i=1}^l F_i}{m})$$

La unión de 2 filtros de bloom A y B puede ser computada aplicando una operación OR:

$$|A \cup B| \approx -\frac{m}{k} \ln(1 - \frac{\sum_{i=1}^m (A \cup B)_i}{m})$$

La intersección de 2 filtros de bloom A y B puede ser computada aplicando una operación AND:

$$|A \cap B| = |A| + |B| - |A \cup B|$$

Consideraciones sobre los filtros de Bloom

- No son una estructura que almacena datos por sí misma, pero puede ser usada como un mecanismo de optimización para mejorar el rendimiento de muchas aplicaciones.
- La tasa de falsos positivos debe ser medida y monitorizada. El rendimiento de los filtros de bloom se puede desplomar si hay demasiados elementos insertados.

Funciones Hash

En teoría, se deben seleccionar k funciones hash diferentes para implementar en los filtros de bloom. En la práctica las funciones hash son generadas por un esquema de doble hasing:

$$h_i(x) = h_1(x) + i \cdot h_2(x)$$

En este caso, dos funciones hash diferentes son requeridas. También es común suar una función hash con valores de entrada divididos en dos partes.

Diccionarios y Descomposición

Muchas palabras pueden ser descompuestas usando múltiples reglas, pero algunas requieren buscar su separación en una tabla de descomposición. Un filtro bloom es usado para almacenar la representación de dicha tabla. Si la palabra no está almacenada en el bloom filter, su descomposición puede ser resuelta usando reglas simples. Si la palabra fue almacenada en el filtro, entonces se ha realizado una búsqueda de tabla. Los falsos positivos significan que la tabla fue buscada cuando no era necesario. Extensiones de esta aplicación pueden ser usadas para sistemas de revisión de pronunciación y diccionarios de contraseñas no válidas y nombres de usuario ya en uso.

Bases de datos

Optimiza operaciones JOIN en sistemas distribuidos. Un nodo A envía un Filtro de Bloom que representa un subconjunto de datos del nodo B. El nodo B devuelve al nodo A pares que han coincidido con el bloom filter. Finalmente, los falsos positivos son eliminados de A. Usando esta aproximación se reducen el consumo de recursos y las comunicaciones.

Proxies de Cacheo distribuidos

Cuando hay un fallo de cache, entonces el proxy intenta adivinar si hay otro proxy que tenga la página web en cache. Intercambiar todo el contenido en cache entre todos los proxies es muy caro. Un filtro de Bloom que representa los contenidos de la cache puede ser usado. Cuando un proxy tiene que encontrar otro que tenga la web en caché, simplemente revisa su Filtro de Bloom. Un falso positivo significa que la petición será hecha al proxy para descubrir que no tiene la página en cache. En esquema, los falsos negativos son posibles ya que la cache pudo haber sido actualizada durante el proceso.

Problemas de seguridad

Los filtros de bloom no son una estructura de datos segura. No pueden ser considerados una medida de seguridad activa. Pueden ser utilizados para intentar preservar la privacidad de los datos que representan hasta cierto punto. Son vulnerables a ciertos tipos de ataques.

- Es fácil manipular y modificar sus contenidos para incrementar los falsos positivos (Ataque de polución)
- El análisis probabilístico usando frecuencias puede resultar en un enlace de registros con bases de datos externas para revelar datos.

El uso de funciones hash criptográficas no reduce el número de falsos positivos. Pueden ser usadas si se desea, pero pueden afectar considerablemente el rendimiento del filtro. Normalmente se usan funciones hash no criptográficas. Si se usa un esquema de doble hasheo basado en slicing, las funciones criptográficas pueden ser una buena elección.

Filtros Bloom para enlace de registros que preservan la privacidad

El enlace de registros empareja registros de diferentes bases de datos que se refieren a la misma entidad, se suelen usar para soportar procesos como la realización de decisiones. Este problema suele surgir en ambientes relacionados con la salud. Se necesita publicar los datos para ayudar a la comunidad a realizar su propio análisis de datos de la mejor manera posible. Para ello se recomienda seguir los siguientes pasos:

1. Preprocesado: Se sanean y estandarizan los datos.
2. Filtrado: Optimiza el proceso de búsqueda reduciendo el espacio de búsqueda.

3. Comparación: Computa una puntuación similar
4. Clasificación: Hace decisiones de acuerdo a la métrica.

Intercambiar información sensible con terceras partes tiene problemas de privacidad. La idea es que la tercera parte sea capaz de enlazar los registros sin revelar la información. Usando codificación clásica o mecanismos de cifrado puede resultar en computación segura muy cara que garantiza una privacidad robusta.

Respuesta Ordinal preservadora de la privacidad aleatoriamente agregable (RAPPOR)

Es un mecanismo para recoger estadísticas anónimamente desde el cliente de software del usuario, garantizando la privacidad y permitiendo el análisis de los datos sobre la información recolectada. Está basado en el mecanismo de respuesta aleatorizada. Este método permite recolectar estadísticas más de una vez preservando la privacidad. Los datos son representados por una cadena de bits que contiene una respuesta aleatorizada a una característica.

- Para valores categóricos, cada bit puede ser representado como la presencia o ausencia de dicha característica
- Para valores numéricos, cada bit puede ser asociado con un predicado para un rango de valores.
- Otras características no categoricas o no numéricas son tratadas con cadenas de caracteres.

Dichos valores son codificados usando filtros bloom y se le añade salt con ruido para hacerlos más robustos contra ataques longitudinales. Este método tiene los siguientes parámetros:

- Cada usuario es asignado a uno de los M grupos distintivos
- Un filtro bloom de tamaño N con K funciones hash se usa para representar los predicados que van a ser reportados al servidor
- Se añade ruido al filtro de bloom creado de acuerdo con 3 parámetros de privacidad: p , q y f

Procedimiento

- Dado un valor v , se hasha en el $FB(B)$ de forma que ciertos bits serán establecidos a 1 y el resto como 0.
- Se añade ruido a B para obtener una Respuesta Permanente Aleatorizada (PRR) B' que será guardada para futuras respuestas para el mismo valor v

$$B'_i = \begin{cases} 1, & \text{con probabilidad } \frac{1}{2}f \ll 0, \\ & \text{con probabilidad } \frac{1}{2}f \ll B_i, \\ & \text{con probabilidad } 1 - f \end{cases}$$

- Cada vez que el cliente envía un reporte al servidor, una respuesta aleatorizada instantánea (IRR) S es construida basada en B' estableciendo cada bit i en S a 1 con probabilidades:

$$P(S_i = 1) = \begin{cases} q, & \text{if } B'_i = 1 \\ p, & \text{if } B'_i = 0 \end{cases}$$

En este procedimiento B nunca se usa para generar un reporte, se usa B' aleatorizada para evitar ataques que puedan quitar el ruido para desenmascarar B . Debido a la forma en la que está construida, B' puede o no contener información sobre B . B' Nunca se envía directamente en los reportes, se le añade ruido previamente en S de forma que localizar al cliente basándose en B' sea más difícil. Los adversarios pueden aprender algo sobre B' con el tiempo pero no van a ser capaces de distinguir entre los bits verdaderos y el ruido, por lo que B no va a ser nunca revelada. A pesar del ruido introducido por este procedimiento, es posible agregar reportes individuales y extraer información útil mediante el uso de análisis estadístico.

Análisis de datos

Como no se reportan datos a través de los filtros de bloom, debemos construir un set candidato compuesto por Z elementos de acuerdo a los datos que estamos recolectando. Para cada uno de los valores en los candidatos debemos identificar cual de los bits será activado cuando se aplican las funciones hash en ellos. Debido a la privacidad diferencial no podemos identificar los sets originalmente establecidos a 1, pero podemos estimar la media de cuanto ruido se ha añadido.

$$t_{i,j} = \frac{c_{i,j}}{N_j} - \left(p + \frac{1}{2} f q - \frac{1}{2} f p \right) \frac{1}{(1-f)(q-p)}$$

Tras eso, se crea un sistema de ecuaciones lineales para computar el conteo de cada elemento en el set de candidatos.

$$X = \begin{bmatrix} X_{i,j,z} & \cdots & X_{i,j,z} \\ \vdots & & \vdots \\ X_{i,j,z} & \cdots & X_{i,j,z} \end{bmatrix}, \quad \text{donde } X_{i,j,z} = \begin{cases} 1, & \text{Si } B_i = 1 \\ \text{para el elemento } z \text{ en el grupo } j \end{cases} \quad \text{En caso contrario } \end{cases}$$

Tenemos que resolver $XZ = Y$ para descubrir los conteos del set candidato Z . Normalmente no podemos resolver directamente dicho sistema de ecuaciones por que puede no tener una solución perfecta al ser Y estimaciones de conteo. Normalmente se usa el método de regresión lineal combinado con algunos test de significancia para descartar candidatos que no tienen suficientes pruebas de los conteos estimados.

Servicio de Geofencing

El Geofencing es un procedimiento que define uno o más estructuras virtuales que corresponden con una zona geográfica real. Dichas áreas son delimitadas por sus ubicaciones a través de uno o más

polígonos de acuerdo a sus coordenadas. Cuando un cliente entra o sale dicha región, una notificación o acción puede ser disparada. Monitorizar la ubicación de la gente es un proceso muy sensible ya que la información sobre su ubicación, actividades y comportamiento pueden ser revelados.

Nos centramos en un sistema diseñado para localizar vehículos que no tienen permitido el acceso a cierta zona. El operador de la flota debe ser notificado de que cierto vehículo ha abandonado el área autorizada. Cuando un vehículo esté dentro de la región autorizada, el operador no debe ser capaz de saber su posición exacta para garantizar la privacidad del cliente. Difundir alguna de las regiones a una tercera parte puede ser potencialmente dañino ya que puede conocer información de los clientes.

Verificación Simple de pago en Bitcoin

El SPV es un método empleado por nodos ligeros dentro de la red bitcoin. Al contrario que los nodos completos, no tienen una copia entera de la blockchain. Simplemente monitorizan algunas transacciones de las que están interesados. Dichas transacciones de interés deben ser listadas a los nodos completos para recibir actualizaciones. Publicar una lista de transacciones de interés sin ninguna medida de privacidad puede llevar a la revelación de datos críticos.

Este problema fue resuelto con filtros de Bloom. Los nodos ligeros crean un filtro de bloom que contiene las direcciones de interés y se lo envía al nodo completo, incluyendo una clave pública y una clave hash pública. La privacidad es proveída por los Filtros de Bloom confía en la posibilidad de obtener falsos positivos. El nodo completo envía actualizaciones para todas las direcciones que coinciden con el filtro, pero no sabes si son falsos positivos o no. El nodo ligero debe descartar las direcciones que pertenecen a los falsos positivos y mantener solo las que le interesa. A pesar de eso, debido a un error de diseño, la tasa de falsos positivos se desploma rápidamente, por lo que no provee mucha privacidad.

Ataque de Reidentificación PPRL

En un procedimiento PPRL se intercambia información sensible, por lo que son susceptibles a ataques para tratar de revelar información. Supongamos que un atacante fuera capaz de obtener algún filtro de bloom con el hashing completamente secreto. Primero, la colección de filtros debe ser analizada para tratar de adivinar que tipo de atributos tiene codificados. El atacante puede usar una BD pública y un set secreto de los valores más frecuentes V de un atributo que coincide con la distribución de los filtros de bloom a atacar.

- Los filtros deben ser alineados con valores de atributos de acuerdo a las frecuencias observadas.
 - Tanto los filtros como los valores son ordenados de acuerdo a sus frecuencias en orden inverso.
 - Los pares cuya frecuencia está por encima de cierto límite m son seleccionados.
- Se analiza de forma separada cada bit de la posición P de cada filtro.
 - Si el bit de la posición P tiene el valor 1, entonces se añaden todos los q -gramos de V_i correspondientes al set candidato $c\{p\}^+$
 - Si la posición del bit P tiene el valor 0, entonces se añaden todos los q -gramos de

V_i correspondientes al set candidato $c\{p\}^+ - c\{p\}^-$

- Una vez todos los pares han sido explorados, el set de candidatos verdaderos para cada posición p del filtro es determinado:

$$c[p] = c[p]^+ / c[p]^-$$

From:

<https://knoppia.net/> - **Knoppia**

Permanent link:

https://knoppia.net/doku.php?id=pan:filtros_bloom_v2&rev=1767881986

Last update: **2026/01/08 14:19**

