

Sistema de Detección de Trending Topics usando Lossy Counting y Ventanas Deslizantes

Rodrigo Silva

2025

Abstract

Este artículo presenta un sistema eficiente para la detección de trending topics en tiempo real utilizando el algoritmo *Lossy Counting* con *Min-Heap* combinado con ventanas deslizantes. El enfoque permite identificar temas emergentes dentro de flujos de texto extensos, adaptándose dinámicamente a cambios temporales mediante podas periódicas, y generando **wordclouds dinámicos** que visualizan la evolución de las tendencias por ventana temporal.

Palabras clave: Lossy Counting, Trending Topics, Wordcloud, Ventanas Deslizantes, Min-Heap, Procesamiento de Texto

1. Introducción

La detección de tendencias en grandes volúmenes de texto, como publicaciones o noticias, requiere sistemas que identifiquen cambios temáticos de manera continua y eficiente. Procesar cada documento desde cero resulta computacionalmente inviable cuando se trabaja con flujos masivos de información, por esta razón, se emplean algoritmos de stream mining que operan con memoria controlada y permiten identificar patrones emergentes sin necesidad de almacenar todo el histórico de datos.

En este trabajo se implementa una arquitectura basada en el algoritmo Lossy Counting optimizado con Min-Heap, combinado con ventanas deslizantes de tamaño k . Esta combinación permite observar cómo cambian los temas predominantes a medida que nuevos textos son incorporados al flujo, manteniendo un balance óptimo entre precisión y eficiencia computacional.

2. Arquitectura del Sistema

Componente	Función	Ventaja
Preprocesador	Limpieza, minúsculas, eliminación de stopwords y tokenización	Estandariza los datos antes del conteo
Lossy Counting	Conteo aproximado con podas periódicas	Adapta dinámicamente a cambios temporales
Min-Heap	Extracción eficiente del Top-K	Evita ordenamiento completo del HashMap
Ventana Deslizante	Procesamiento de los últimos k textos	Captura la evolución de temas en el tiempo
Wordcloud Dinámico	Visualización de los Top-K tokens por ventana	Muestra la evolución temática

2.1. Flujo del Proceso

TXT → Preprocesamiento → Lossy Counting + Heap → Top-K → Wordcloud

3. Metodología

3.1. Preprocesamiento de Texto

Antes de contar palabras, cada documento pasa por una limpieza estandarizada. Por ejemplo, el texto “El partido de fútbol estuvo INCREÍBLE! #Deporte” se transforma a minúsculas, se eliminan símbolos especiales y stopwords, se aplica lematización y finalmente se tokeniza en [“partido”, “fútbol”, “increíble”, “deporte”].

4. Algoritmo Lossy Counting

Lossy Counting mantiene un HashMap donde cada palabra tiene dos valores: su frecuencia y un error estimado. La idea central es procesar tokens en “buckets” (grupos) y cada cierto número de tokens ejecutar una “poda” que elimina palabras poco frecuentes.

4.1. Ejemplo Práctico

Supongamos que configuramos el sistema para hacer una poda cada 5 tokens (bucket_size = 5). Procesaremos la siguiente secuencia de noticias:

Tokens de entrada:

```
[fútbol, gol, fútbol, partido, gol, fútbol, terremoto,
terremoto, gol, alerta]
```

4.1.1. Fase 1: Procesando tokens 1-5

```
Token 1: "fútbol" → HashMap = {fútbol: (1, 0)}
Token 2: "gol" → HashMap = {fútbol: (1, 0), gol: (1, 0)}
Token 3: "fútbol" → HashMap = {fútbol: (2, 0), gol: (1, 0)}
Token 4: "partido" → HashMap = {fútbol: (2, 0), gol: (1, 0),
partido: (1, 0)}
Token 5: "gol" → HashMap = {fútbol: (2, 0), gol: (2, 0),
partido: (1, 0)}
```

¡Llegamos a 5 tokens! Ejecutamos PODA número 1:

La regla de poda dice: eliminar palabras donde (frecuencia + error) sea menor o igual al número de poda. Como es la poda número 1, verificamos si (frecuencia + error) ≤ 1 :

- fútbol: $(2 + 0) = 2 > 1 \rightarrow$ SE MANTIENE
- gol: $(2 + 0) = 2 > 1 \rightarrow$ SE MANTIENE
- partido: $(1 + 0) = 1 \leq 1 \rightarrow$ SE ELIMINA

```
HashMap después de poda 1 = {fútbol: (2, 0), gol: (2, 0)}
```

4.1.2. Fase 2: Procesando tokens 6-10

```
Token 6: "fútbol" → HashMap = {fútbol: (3, 0), gol: (2, 0)}
Token 7: "terremoto" → HashMap = {fútbol: (3, 0), gol: (2, 0),
terremoto: (1, 1)}
(notas: terremoto entra con error=1 porque
es nuevo después de 1 poda)
Token 8: "terremoto" → HashMap = {fútbol: (3, 0), gol: (2, 0),
terremoto: (2, 1)}
Token 9: "gol" → HashMap = {fútbol: (3, 0), gol: (3, 0),
terremoto: (2, 1)}
Token 10: "alerta" → HashMap = {fútbol: (3, 0), gol: (3, 0),
terremoto: (2, 1), alerta: (1, 1)}
```

¡Llegamos a 10 tokens! Ejecutamos PODA número 2:

Ahora verificamos si (frecuencia + error) ≤ 2 :

- fútbol: $(3 + 0) = 3 > 2 \rightarrow$ SE MANTIENE
- gol: $(3 + 0) = 3 > 2 \rightarrow$ SE MANTIENE
- terremoto: $(2 + 1) = 3 > 2 \rightarrow$ SE MANTIENE
- alerta: $(1 + 1) = 2 \leq 2 \rightarrow$ SE ELIMINA

```
HashMap final = {fútbol: (3, 0), gol: (3, 0), terremoto: (2, 1)}
```

4.2. Resultado del Proceso

Al finalizar, nuestro HashMap contiene solo 3 palabras en lugar de las 5 únicas originales. Las palabras “partido” y “alerta” fueron eliminadas por aparecer pocas veces. Las palabras más frecuentes (“fútbol”, “gol”, “terremoto”) sobrevivieron todas las podas.

Este mecanismo es clave: en un flujo real de miles de tokens, las palabras verdaderamente importantes (trending topics) aparecerán muchas veces y sobrevivirán todas las podas, mientras que palabras raras o de temas antiguos serán eliminadas automáticamente.

5. Integración del Min-Heap

Una vez que tenemos el HashMap con las palabras frecuentes, necesitamos extraer el Top-K (por ejemplo, las 10 más frecuentes). En lugar de ordenar todo el HashMap, usamos un Min-Heap de tamaño K.

El proceso es simple: recorremos el HashMap una vez. Para cada palabra, si el heap tiene menos de K elementos, la insertamos. Si el heap ya está lleno, comparamos la frecuencia de la palabra actual con la mínima del heap: si es mayor, sacamos el mínimo e insertamos la nueva palabra. Al final, el heap contiene exactamente las K palabras más frecuentes.

En nuestro ejemplo con K=3, el heap final contendría: fútbol (3), gol (3), terremoto (2).

6. Ventanas Deslizantes

El procesamiento mediante ventanas deslizantes captura la evolución temporal de tendencias. Cada ventana agrupa k documentos consecutivos, procesa todos sus tokens mediante Lossy Counting, ejecuta poda, extrae el Top-K con Min-Heap, y genera un wordcloud. Posteriormente la ventana avanza un documento: el texto más antiguo sale del contexto y un nuevo texto ingresa, permitiendo observar cómo emergen y desvanecen diferentes temas.

Consideremos un corpus de 8 documentos con $k = 3$. La primera ventana analiza documentos 1-3, donde predominan términos deportivos como “fútbol”, “gol” y “partido”. El wordcloud resultante visualiza esta dominancia temática con tamaños proporcionales a frecuencia. La segunda ventana desliza a documentos 2-4: el documento 1 sale mientras ingresa el 4, que introduce términos como “terremoto” y “alerta”. El HashMap ahora combina remanentes del tema deportivo con el emergente tema sismológico. Tras la poda, términos deportivos menos frecuentes son eliminados mientras “terremoto” asciende rápidamente en el ranking. La tercera ventana (documentos 3-5) completa la transición: el tema deportivo ha desaparecido completamente del Top-K, reemplazado por un vocabulario dominado por “terremoto”, “alerta” y “magnitud”.

Esta evolución demuestra la capacidad del sistema para adaptarse dinámicamente. Las podas periódicas actúan como mecanismo de olvido, reduciendo gradualmente conteos de términos que dejan de aparecer. Simultáneamente, nuevos términos frecuentes ascienden rápidamente sin verse penalizados por conteos históricos de temas anteriores, un problema que afectaría a estructuras de memoria fija como Space-Saving.

7. Análisis de Rendimiento

La complejidad temporal por ventana se descompone en componentes claramente identificables. El preprocesamiento de W tokens totales requiere $O(W)$ operaciones para limpieza, normalización y tokenización. La inserción en HashMap promedia $O(1)$ por token, totalizando $O(W)$. Las podas ejecutadas periódicamente tienen costo amortizado de $O(W)$. La construcción del Min-Heap requiere $O(M \log K)$ y la extracción final $O(K \log K)$. Sumando todos los componentes, la complejidad total es $O(W + M \log K)$ donde M es el tamaño del HashMap después de podas.

8. Referencias

1. Manku, G. S., & Motwani, R. Approximate Frequency Counts over Data Streams. *VLDB*
2. <https://www.vldb.org/conf/2002/S10P03.pdf>