Corso di laurea magistrale in Scienze Statistiche
Metodi informatici per statistica e data science
2023/2024

Università degli Studi di Padova

Individuazione dei duplicati nei dati di flusso

Matteo C.
Michela R.

Indice della presentazione

- Problema
- Dati
- Metodi utilizzati
- Efficienza ed efficacia
- Esperimenti
- Conclusioni

Contesto

- Flusso continuo e potenzialmente infinito
- Memoria limitata per il salvataggio dei dati
- Presenza di duplicati

→ Rilevamento esatto dei duplicati non fattibile

Contesto applicativo

Web crawling: attività automatica di scansione ed indicizzazione delle pagine web.

Obiettivo: individuazione efficiente dei duplicati (URL già visitati)

Dataset di riferimento

Common Crawl: organizzazione no-profit fondata nel 2007 che gestisce un archivio gratuito e aperto di dati di scansione web:

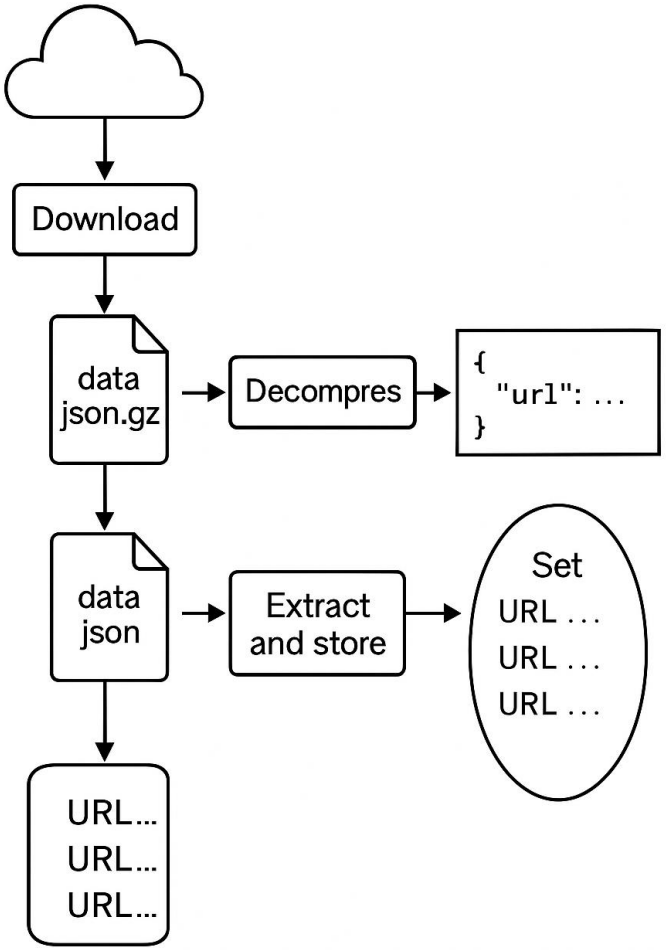
- 250 mld di pagine web raccolte in 18 anni
- Aggiornato con 2-5 mld di pagine ogni mese
- Citato in oltre 10 k articoli di ricerca

Dataset di riferimento

https://data.commoncrawl.org/cc-index/collections/index.html

ID	Announcement	Billion Pages	Index File Listing
CC-MAIN-2025-08	February 2025	2.67	CC-MAIN-2025-08/cc-index.paths.gz
CC-MAIN-2025-05	January 2025	3.00	CC-MAIN-2025-05/cc-index.paths.gz
CC-MAIN-2024-51	December 2024	2.64	CC-MAIN-2024-51/cc-index.paths.gz
CC-MAIN-2024-46	November 2024	2.68	CC-MAIN-2024-46/cc-index.paths.gz
CC-MAIN-2024-42	October 2024	2.49	CC-MAIN-2024-42/cc-index.paths.gz
CC-MAIN-2024-38	September 2024	2.80	CC-MAIN-2024-38/cc-index.paths.gz
CC-MAIN-2024-33	August 2024	2.30	CC-MAIN-2024-33/cc-index.paths.gz
CC-MAIN-2024-30	July 2024	2.50	CC-MAIN-2024-30/cc-index.paths.gz
CC-MAIN-2024-26	June 2024	2.70	CC-MAIN-2024-26/cc-index.paths.gz
CC-MAIN-2024-22	May 2024	2.70	CC-MAIN-2024-22/cc-index.paths.gz
CC-MAIN-2024-18	April 2024	2.70	CC-MAIN-2024-18/cc-index.paths.gz

Decompressione dei dati



1) Download da Common Crawl (.gzip)
2) Decompressione *parsing* riga per riga degli oggetti JSON
3) Estrazione dell'URL
4) Inserimento in un set

Salvataggio dei dati: DuckBD

Si tratta di un database ***in-process***:

- Integrato in Python;
- Portabile (file .duckdb);
- Supportato da SQL per interrogazioni complesse ed analisi;
- Indicizzato;
- Nessuna configurazione di porte, utenti, permessi, ecc.;
- No sovraccarichi dovuti a comunicazioni client-server.

Salvataggio dei dati: problema vs soluzione

File.txt	File.duckdb
Formato testuale (ASCII o UTF-8)	Formato binario ottimizzato
Nessuna decompressione	Compressione colonnare
Assenza di struttura	Supporto del tipo di dati (int, char,…)
Strumenti di analisi da implementare esternamente con conseguente aumento della complessità	Compatibilità con SQL
Necessità di lettura sequenziale del file completo	Sfruttamento dell'indicizzazione e possibilità di caricamento parziale dei dati
Formato libero con rischio di errori	Formato strutturato con vincoli e controlli sulla coerenza dei dati

Bloom Filter

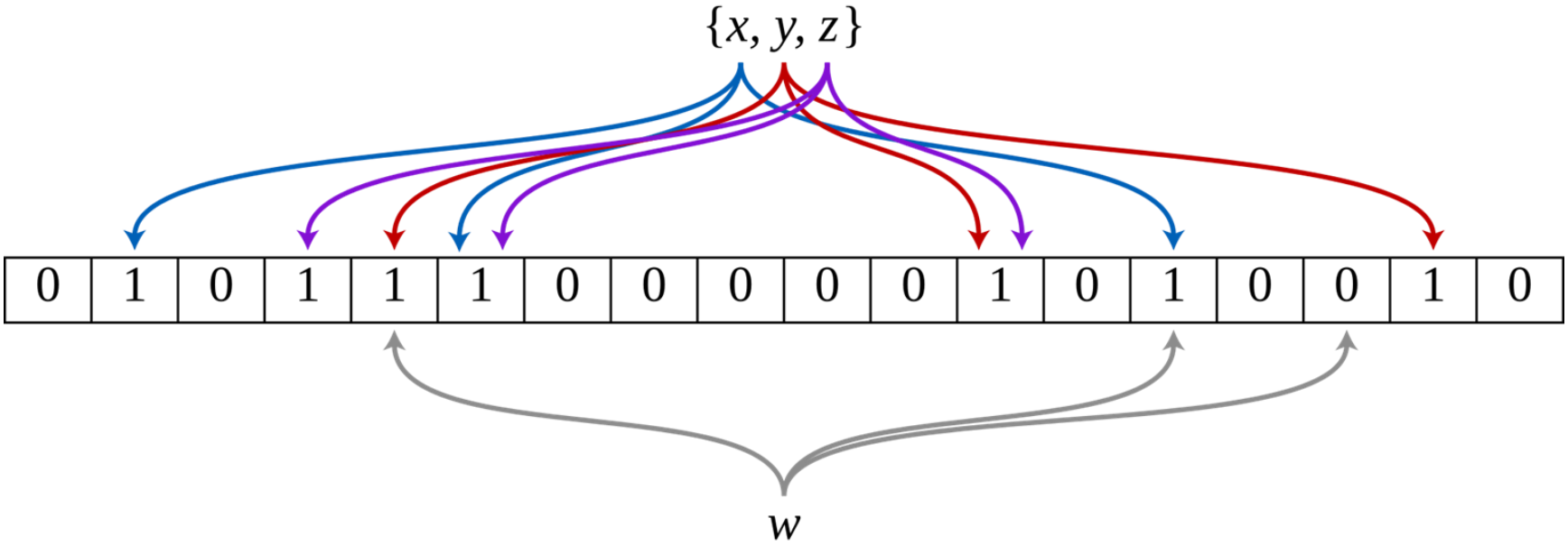
Parametri:

- N: numero di elementi nel flusso di input
- M: spazio totale disponibile (in bit)
- m: numero di celle del filtro (= M)
- K: numero di funzioni hash

Bloom Filter

Inizializzo $SBF[1] \dots SBF[m] = 0$
for ogni $x_i \in S$ **do**:
    for k celle $SBF[h_1(x_i)] \dots SBF[h_k(x_i)]$:
    if tutte le k celle sono uguali a 1 **then**:
        $DuplicateFlag = 1$
    else:
        $DuplicateFlag = 0$
        for ogni cella in $\{SBF[h_1(x_i)] \dots SBF[h_k(x_i)]\}$: **do**:
            $SBF[h(x_i)] = 1$
    output $DuplicateFlag$

Bloom Filter



Bloom Filter: *double hashing*

$$h_i = h_1 + i * h_2, \qquad i = 3, \ldots, k$$

- Si evitano k chiamate indipendenti alle funzioni di hash
- Utilizzo di algoritmi crittografici robusti: MD5 (128 bit) e SHA1 (160 bit)
- Derivazione degli indici del filtro di bassa complessità computazionale:

$$\mathrm{h}_i \,\%\, m$$

Limiti relativi al Bloom Filter

1. Il numero di celle pari a 0 diminuisce in maniera monotona con l'arrivo di nuovi elementi → saturazione della memoria
2. Non viene data importanza agli elementi più recenti (data decay)

→ inadatto a scenari con flussi di dati continui

Soluzione: Stable Bloom Filter

1. Mantiene stabilità nel tasso di Falsi Positivi (FP)
2. Introduce un meccanismo di decadimento controllato

→ adatto a scenari con flussi di dati continui

Stable Bloom Filter

Parametri:

- N: numero di elementi nel flusso di input
- M: spazio totale disponibile in bit
- Max: valore di impostazione delle celle
- d: bit necessari per la rappresentazione di Max
- m: numero di celle del filtro $\left(= \frac{M}{d}\right)$
- K: numero di funzioni hash
- P: numero di celle da decrementare
- FPR: rapporto di Falsi Positivi

Se Max = 1 e P = 0 $\rightarrow$ Bloom Filter.

Considerazioni su P

È il numero di celle casuali decrementate per:

- Evitare il riempimento del filtro
- Mantenere stabili i FP durante l'aggiornamento

$$Inner - term = \left(1 - FP^{1/k}\right)^{1/Max}$$

$$Den_p = \left(\frac{1}{Inner - term} - 1\right)\left(\frac{1}{k} - \frac{1}{m}\right)$$

$$P = (Den_p)^{-1}$$

Stable Bloom Filter

Inizializzo $SBF[1] \dots SBF[m] = 0$

for ogni $x_i \in S$ **do**:

for k celle $SBF[h_1(x_i)] \dots SBF[h_k(x_i)]$:

if nessuna delle k celle sopra è 0 **then**:

$DuplicateFlag = 1$

else:

$DuplicateFlag = 0$

seleziona p differenti celle randomiche

$SBF[j_1] \dots SBF[j_p], p \in \{1, \dots, m\}$:

for $SBF[j] \in \{SBF[j_1] \dots SBF[j_p]\}$ **do**:

if $SBF[j] >= 1$ **then**:

$SBF[j] = SBF[j] - 1$

for ogni cella in $\{SBF[h_1(x_i)] \dots SBF[h_k(x_i)]\}$: **do**:

$SBF[h(x_i)] = Max$

output $DuplicateFlag$

Efficienza e complessità

Bloom Filter:

Temporale:

- Inserimento O(k)
- Verifica O(k)

Spaziale:

- Inizializzazione O(M)
- Memoria occupata O(M)

Stable Bloom Filter:

Temporale:

- Inserimento O(k + p)
- Verifica O(k)

Spaziale:

- Inizializzazione O(M)
- Memoria occupata O(M)

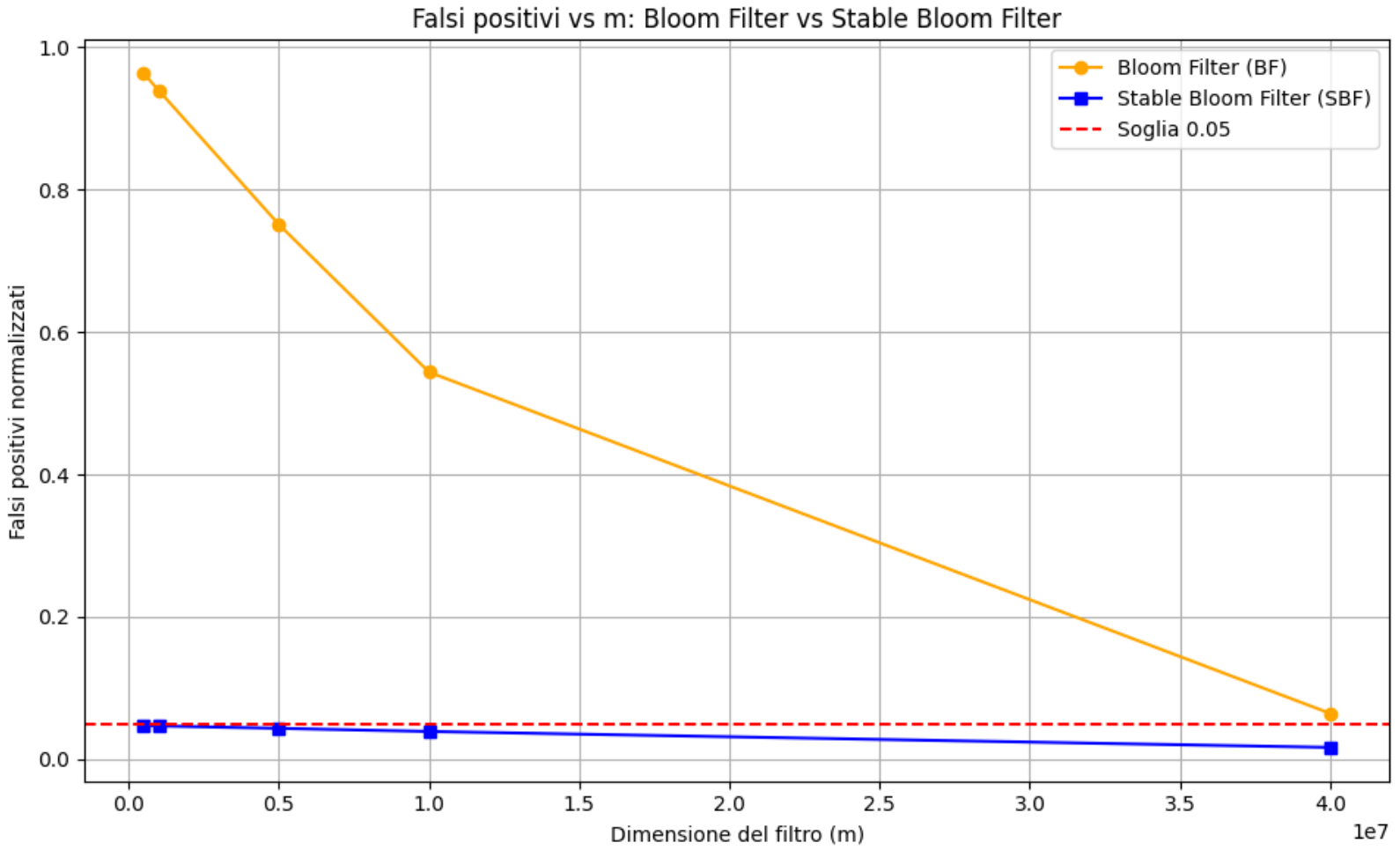
Esperimenti: scelta di m

	Impatto	BF	SBF
m basso	- Memoria + Collisioni e FP	FP → 1 Saturazione rapida	FP costante Minore memoria storica
m alto	+ Memoria - Collisioni e FP	FP inferiori Maggiore precisione	FP costante Maggiore stabilità

Valori empirici:

$m \in \{500\text{ k}, 1\text{ mln}, 5\text{ mln}, 10\text{ mln}, 40\text{ mln}\}$

Esperimenti: scelta di m



Esperimenti: scelta di K

	BF	SBF
K basso	Collisione	Minore stabilità
K alto	• Saturazione rapida del filtro • Aumento del costo computazionale	• Aumento del costo computazionale • Aumento dei Falsi Negativi (FN)
Compromesso	K = ln(2)(m/n)	K con F1 più elevato

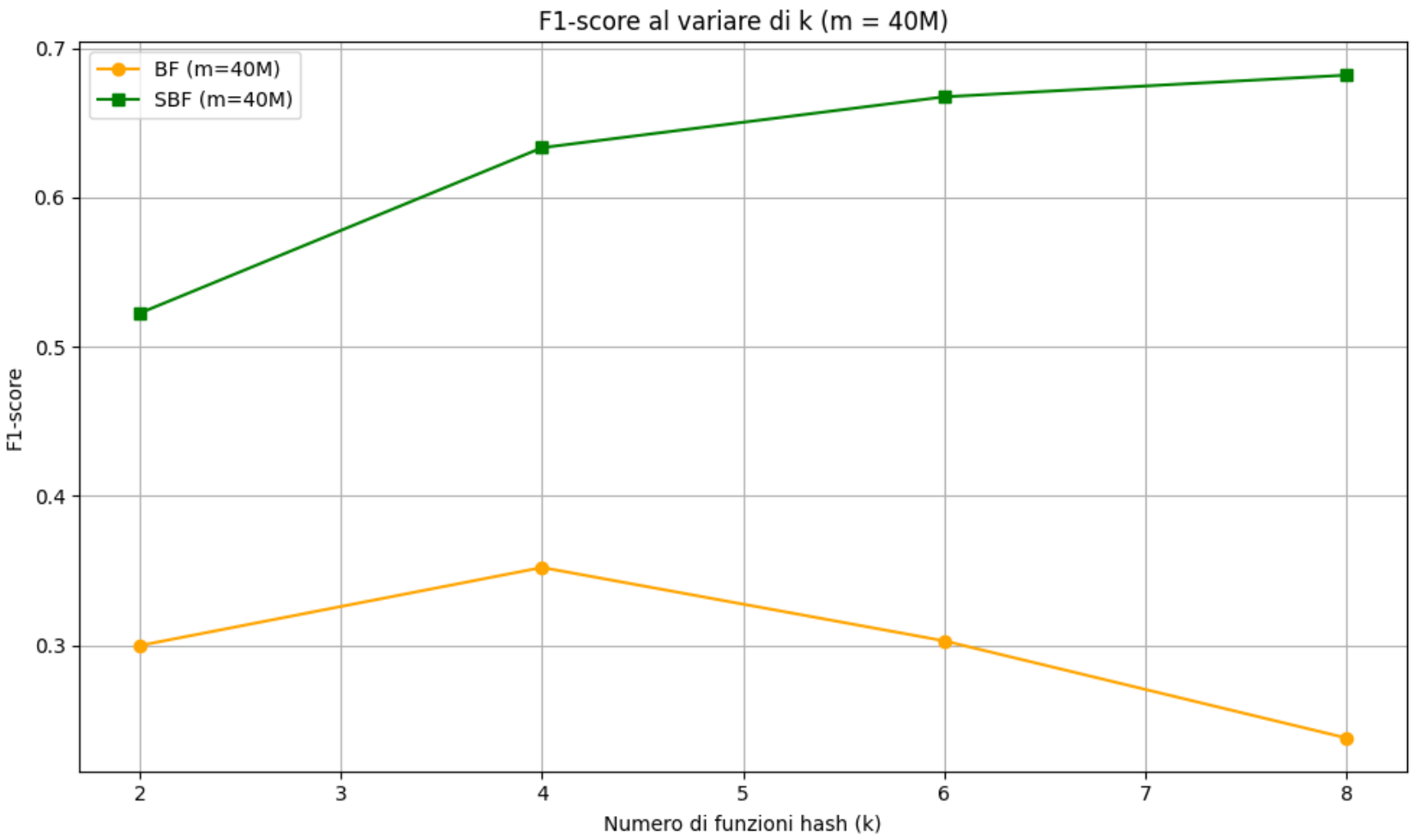
Efficacia

$$F1 = \frac{2TP}{2TP + FP + FN} \epsilon [0; 1]$$

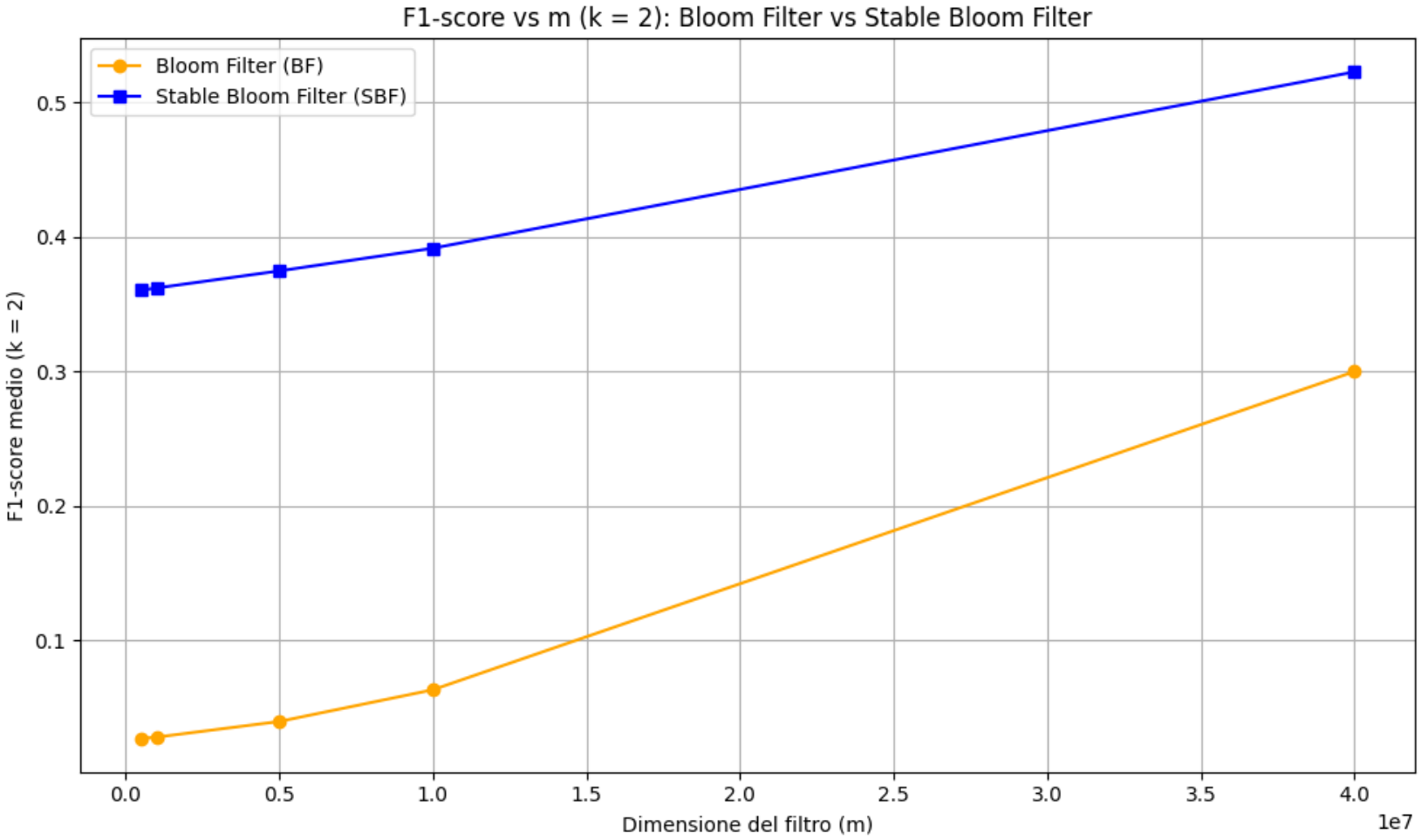
TP = True Positive
FP = False Positive
FN = False Negative

È una misura di accuratezza che bilancia precisione e recall attraverso la loro media armonica.

Scelta di K basta sull'F1



Efficacia

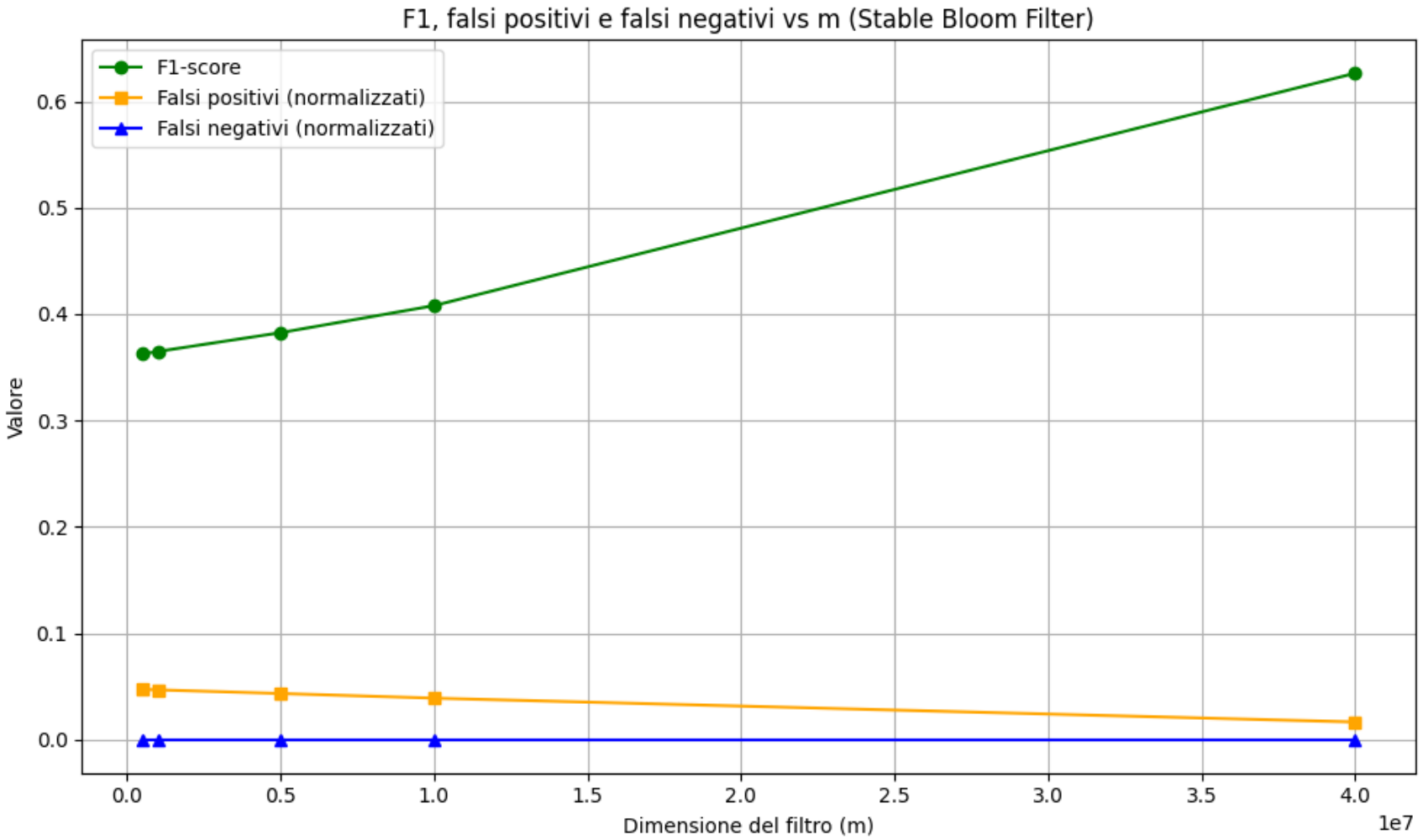

F1-score vs m (k = 2): Bloom Filter vs Stable Bloom Filter
Bloom Filter (BF)
Stable Bloom Filter (SBF)
F1-score medio (k = 2)
0.1
0.2
0.3
0.4
0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
1e7
Dimensione del filtro (m)

Deterioramento dell'F1

Per flussi molto estesi (come nel presente caso) si osserva un F1 inferiore a quello atteso a causa di:

- Distribuzione non uniforme dei dati (cluster di URL o picchi improvvisi di duplicati)
- Numero elevato elementi distinti

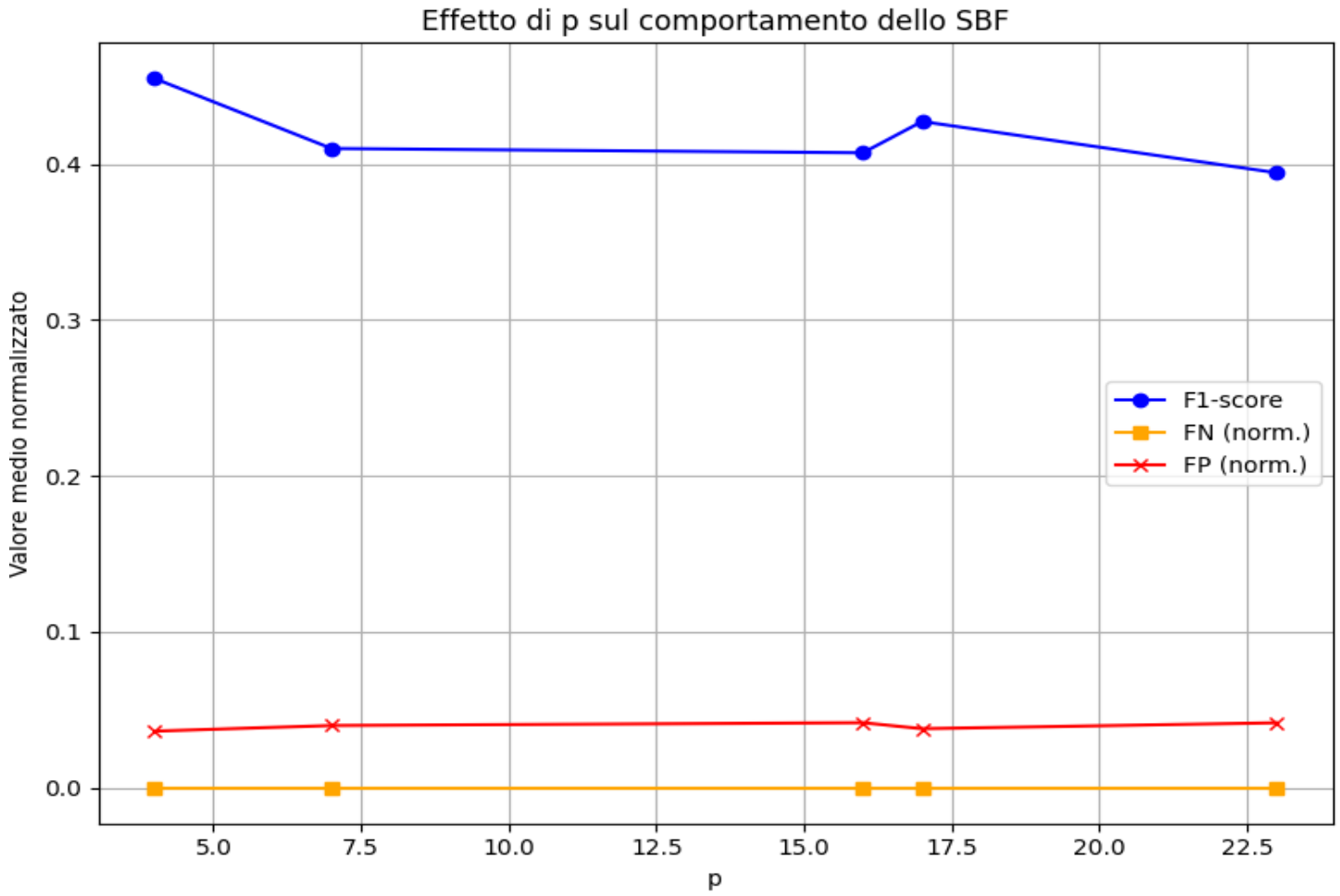
Deterioramento dell'F1



Considerazioni su P: una coperta troppo corta

- P troppo basso → tutti i contatori al massimo, saturazione del filtro
- P troppo alto → filtro troppo volatile, perdita eccessiva di informazione

Considerazioni su P: una coperta troppo corta



Conclusioni

Combinazioni migliori di parametri in termini di F1:

Fps	m	K	Max	p	F1	Tempo (s)
0,05	40 mln	8	1	4	0,748	136,5
0,05	40 mln	6	1	4	0,730	123,5
0,05	40 mln	4	1	4	0,666	103,1

- La prima configurazione raggiunge l'F1 più elevato
- È stato scelto Max = 1 coerentemente con quanto raccomandato nella letteratura, per rendere il comportamento dello Stable Bloom Filter paragonabile al Bloom Filter classico in termini di aggiornamento binario.

Conclusioni

È stato implementato lo Stable Bloom Filter (SBF) per la rilevazione approssimata dei duplicati in flussi di dati.

I confronti sperimentali con il Bloom Filter classico mostrano che, in questi scenari, lo Stable Bloom Filter:

- Ottiene uno score F1 superiore
- Gestisce dinamicamente la memoria grazie ad un decadimento controllato.

Bibliografia

- J. Leskovec, A. Rajaraman, J. D. Ullman, Mining of Massive Datasets, 2014
- Fan Deng and Davood Rafiei, Approximately Detecting Duplicates for Streaming Data using Stable Bloom Filters, 2006
- https://commoncrawl.org/blog/october-2024-crawl-archive-now-available
- https://commoncrawl.org/blog/announcing-the-common-crawl-index
- codice BF: https://www.w3resource.com/python-exercises/advanced/python-bloom-filter-implementation.php
- hashing: https://dl.acm.org/ A. Kirsch e M. Mitzenmacher
- DuckDB: https://duckdb.org/2024/07/09/memory-management.html