



Mini-progetto

LSH near duplicate document detection

Pietro Stangherlin 2129207

Nicola Castelletti 2109302

2023/2024

Indice della presentazione

- ❑ Obiettivi
- ❑ Metodi utilizzati
- ❑ Esperimenti
- ❑ Conclusioni

Background

Problema: in grandi raccolte di documenti (in particolare testuali) è possibile che vi siano dei documenti duplicati, quasi uguali, oppure che un documento sia già contenuto in un altro; è inoltre possibile che tali documenti abbiano identificativi diversi o comunque non direttamente riconducibili tra loro.

Degli esempi possono essere:

- medesimi documenti scannerizzati più volte da entità diverse.
- un'organizzazione che mantiene versioni diverse, ovvero con lievi modifiche, di uno stesso documento.

Obbiettivi

Sviluppare un metodo basato su MinHash e LSH per identificare i documenti quasi-duplicati.

Valutare l'efficacia e analizzare l'efficienza
-> Specifica misure di efficacia

Metodi utilizzati

- ❑ Creazione della collezione sperimentale
- ❑ Shingling
- ❑ MinHash
- ❑ LSH

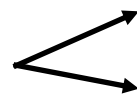
Creazione collezione sperimentale (1)

- 1) Si assume di avere una collezione iniziale senza duplicati e quasi – duplicati. Ogni documento presenta un codice identificativo (id).
- 2) Si seleziona (ad esempio tramite un generatore di numeri casuali) un sottoinsieme di documenti da cui generare i quasi duplicati.
- 3) Scelta del tipo di modifica dei documenti per la creazione dei semi duplicati: sostituzione di alcuni caratteri con altri (random), per simulare un risultato di un programma di riconoscimento ottico di testo. Parametro relativo alla frequenza di sostituzione (bassa o elevata)

Creazione collezione sperimentale (2)

4) A ogni quasi-duplicato si assegna un id univoco (id1) e l'id del documento da cui è stato generato (id2). Viene creata una nuova collezione con i quasi-duplicati e i documenti originali (per questi ultimi il campo l'id2 è posto uguale a None)

Id	text
1	hello world
2	Bob, Alice and Ulm
3	nosce te ipsum



id1	id2	text
1	None	hello world
2	None	Bob, Alice and Ulm
3	None	nosce te ipsum
4	3	n@sCe t3 ipsuN

Creazione collezione sperimentale (3)

4) Durante la creazione della nuova collezione si crea una struttura dati (ad esempio dizionario) o file in cui per ogni id di un documento originale sono associati gli id dei documenti artificiali (o viceversa), in questo modo la procedura di valutazione è più efficiente.

Shingling (1)

- ❑ Dimensione delle shingles: $w = 9$ [5]
- ❑ Trattamento caratteri speciali (spazio, newline etc)
- ❑ Salvataggio delle shingles, proposte di [1]:
 - Salvare anche la frequenza delle shingles nel documento (o usare una shingle più volte) -> possibilmente confrontare
 - Salvare solo le shingles uniche (maggiore efficienza)
- ❑ Hash per le shingles (quasi necessaria per ridurre memoria occupata dalla signature): compromesso tra memoria (definizione del numero di bit dell'hash) impiegata e numero di collisioni per shingles diverse. [1] consiglia di impiegare la Rabin Fingerprint [7]. Considerare anche il Rolling Hashing [8] (concetto simile a calcolo di una media mobile)

Shingling (2)

- ❑ Struttura dati per salvare le shingles (i loro hash): l'insieme delle shingles è temporaneo, serve solo per calcolare la signature, poi è eliminato
 - Lista:
 - Contro: occupa più memoria del necessario per shingle ripetute
 - Contro: le shingle ripetute causano delle inutili operazioni in più nel calcolo delle signatures ()
 - Hash set
 - Pro: shingle ripetute sono considerate una sola volta
 - Contro: affinché non vi siano eccessive collisioni tra le chiavi una parte di memoria dell hash set deve essere vuota, tuttavia, ad eccezione di test di dimensioni fuori scala ciò non dovrebbe essere un problema

MinHash (1)

- ❑ Valutare se le signatures di tutti i documenti possono risiedere in memoria centrale, altrimenti
 - Scrittura su disco
 - Modificare parametri delle shingles e delle permutazioni
- ❑ Scelta del numero k di permutazioni e relative funzioni hash. [6] Universal hashing.
- ❑ Nella costruzione del MinHash: valutare se tenere in memoria un dizionario con valori di vettori di k vettori (tanti quante sono le funzioni hash di permutazione) oppure mantenere in memoria le shingles e fare un dizionario alla volta

MinHash (2)

- ❑ Scelta della struttura dati per le signatures: ciascun documento deve essere identificabile tramite id (per i confronti tra bande di signatures dopo LSH).
- ❑ In una possibile implementazione (memory wise) non vengono mantenute in memoria le signatures: ciascuna signature è impiegata per eseguire LSH e poi è rimossa dalla memoria (in questo caso ci si aspetta una riduzione della precisione poiché l'unico criterio di similarità è basato sui bucket LSH)

MinHash (3)

Hashing universale[6].

$h : U \rightarrow [m]$ (bins).

Famiglia $H: \{h: U \rightarrow [m]\}$ è detta k -universale se per ogni $x \neq y$ in U si ha $|\{h \in H: h(x) = h(y)\}| \leq |H|/(m^k)$.

Una semplice implementazione da [6] è la famiglia di funzioni

$$h(x) = [(a \cdot x + b) \bmod p] \bmod m$$

Con a e b interi estratti casualmente e p numero primo.

La presenza di operazioni modulo rende però il calcolo non immediato.

Possibilità:

- calcolare l'hash tramite uno script in C o C++
- impiegare dei «trucchi» computazionali

- ❑ Numero di bande
- ❑ Funzioni hash per la bande
- ❑ Struttura dati per i bucket: hash-table dove ad ogni chiave è associata una lista con gli id dei documenti
- ❑ Diverse implementazioni:
 - in memoria centrale
 - Hash-table
 - Hash-tree o b-tree
 - In memoria di massa

- ❑ Elaborando documento per documento di devono mantenere in memoria gli insiemi di bucket per tutte le bande: ciò può essere problematico. L'alternativa è elaborare banda per banda (mantenendo quindi un solo insieme di bucket in memoria alla volta), ma ciò significa mantenere in memoria tutte le signature e dover passare b (numero di bande) volte sugli n documenti, azione che appare eccessivamente dispendiosa.

Esperimenti

- ❑ Prima esecuzione della procedura sulla collezione originale (senza semi-duplicati artificiali) per tarare i parametri e valutare il massimo grado di similarità presente tra documenti della collezione originale
- ❑ Esecuzioni successive con diversi livelli di modifica dei documenti semi-duplicati artificiali

Conclusioni

Bibliografia

1. A.Z. Broder, “On the resemblance and containment of documents,” Proc. Compression and Complexity of Sequences, pp. 21–29, Positano Italy, 1997
2. A.Z. Broder, M. Charikar, A.M. Frieze, and M. Mitzenmacher, “Min-wise independent permutations,” ACM Symposium on Theory of Computing, pp. 327–336, 1998.
3. P. Indyk and R. Motwani. “Approximate nearest neighbor: towards removing the curse of dimensionality,” ACM Symposium on Theory of Computing, pp. 604–613, 1998.
4. A. Gionis, P. Indyk, and R. Motwani, “Similarity search in high dimensions via hashing,” Proc. Intl. Conf. on Very Large Databases, pp. 518–529, 1999
5. J. Leskovec, A. Rajaraman, and J. Ullman. “Mining of Massive Datasets” Cambridge University Press, Second edition, (2014)

Bibliografia

6. J. Lawrence Carter, Mark N. Wegman, Universal classes of hash functions, Journal of Computer and System Sciences, Volume 18, Issue 2, 1979, Pages 143-154.
7. Michael O. Rabin (1981). "Fingerprinting by Random Polynomials. Center for Research in Computing Technology, Harvard University.
8. Jonathan D. Cohen. 1997. Recursive hashing functions for n-grams. ACM Trans. Inf. Syst. 15, 3 (July 1997), 291–320.