Finding Similar Items

Overview

- Introduction
- Shingling
- MinHashing
- Locality Sensitive Hashing (LSH)

Introduction

A Common Metaphor

- **Many problems can be expressed as finding "similar" sets:**
  - **Find near-neighbors in <u>high-dimensional</u> space**
- **Examples:**
  - **Pages with similar words**
    - For duplicate detection (plagiarism, mirrors), classification by topic
  - **Customers who purchased similar products**
    - Products with similar customer sets (Reccommendation)
  - **Images with similar features**
    - Users who visited similar websites

A frequent issue

Given O objects, described with a set of **d** features, the goal is to find groups of *similar objects*

Objects = Users, pages, tweets, products, trajectories,...

Features = measurable characteristics, such as binary, categorical or real

Similarity(o1,o2) is a function that, taken the set of features of users u1 and u2, returns a value in [0,1]

Operations should be supported fast => without scanning the whole dataset (query), or take almost linear time (clustering)

Example #1: Users

Here O = U users

Features = Personal data, preferences (restaurants, books, products,...), navigational/search behavior,...

	Brahma Bull	Spaghetti House	Mango	Il Fornaio	Zao	Ming's	Ramona's	Straits	Homma's
Alice		Yes	No	Yes				No	
Bob		Yes				No		No	
Cindy				Yes	No			No	
Dave	No			No	Yes	Yes			Yes
Estie				No	Yes	Yes		Yes	
Fred	No						No		

Find users similar to Dave

Cluster similar users

Similarity = Hamming distance

Hamming distance

- Hamming distance measures the number of positions where two equal-length sequences differ.
- Example (Binary):

A = 1011101

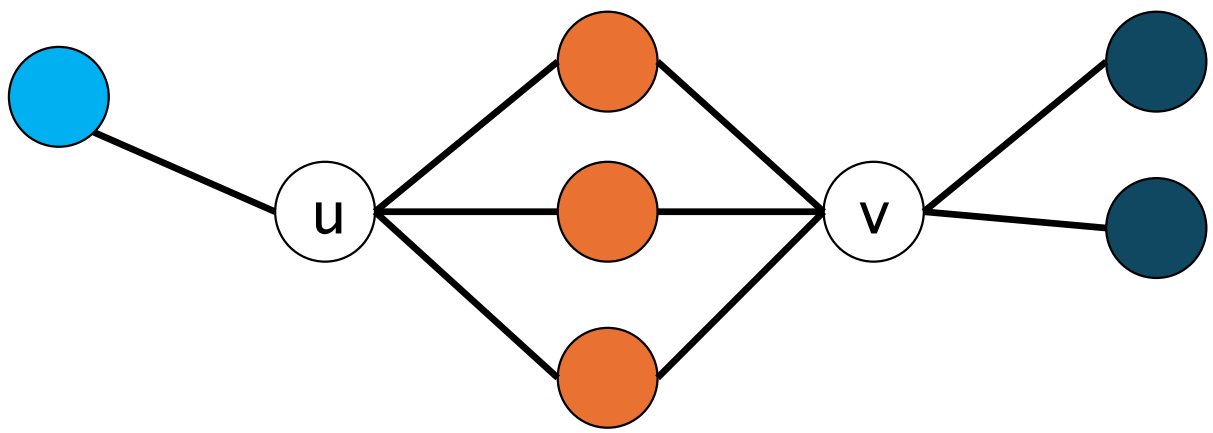
B = 1001001

Hamming distance = 2

Example #2: Graph

Here O = G(V,E) graph

Features = adjacency lists, very sparse



Find nodes similar to u

Cluster similar nodes

Jaccard coeff $\frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|}$

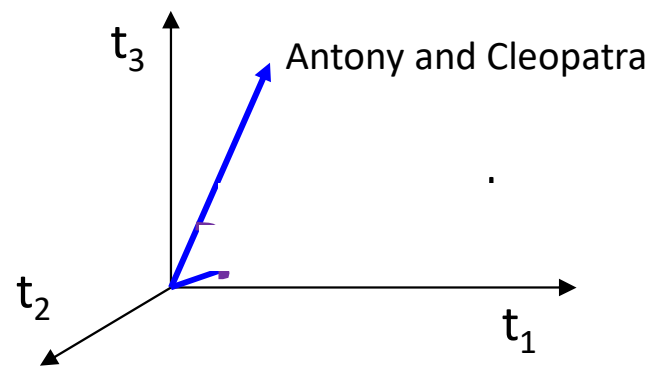
Example #3: Documents

Here O = documents

Features = tf-idf of words, **d is very very large**

	Antony and Cleopatra
Antony	13,1
Brutus	3,0
Caesar	2,3
Calpurnia	0,0
Cleopatra	17,7
mercy	0,5
worser	1,2

Vector Space model



Find documents similar to query

Cluster similar documents

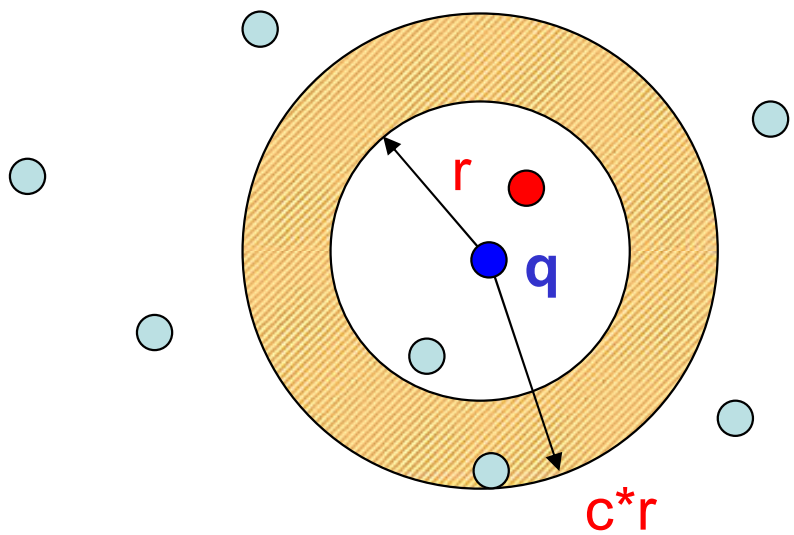
Cosine similarity =

dot product between two document vectors

Example #4: geometry

Here O = d-dimensional points

Features = coordinates, **possibly real**



Find (r, c)-Near Neighbor (r, c)-NN: Given a query point q, find a point p such that $d(q,p) \le c \cdot r$

- Euclidean distance or other norms

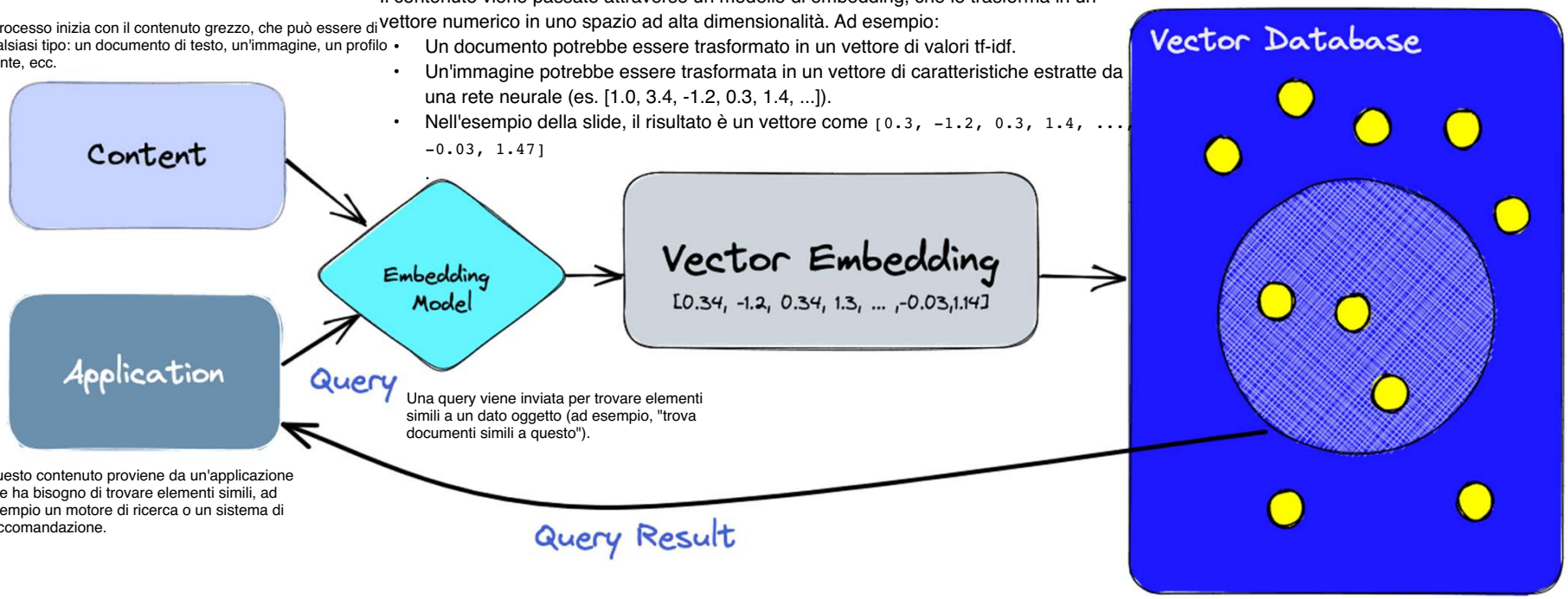
Vector space embedding and LSH

I vettori generati dal modello di embedding vengono memorizzati in un database vettoriale. Questo database contiene tutti gli oggetti rappresentati come vettori (i puntini gialli nella figura rappresentano i vettori nel database).

Il processo inizia con il contenuto grezzo, che può essere di qualsiasi tipo: un documento di testo, un'immagine, un profilo utente, ecc.

Il contenuto viene passato attraverso un modello di embedding, che lo trasforma in un vettore numerico in uno spazio ad alta dimensionalità. Ad esempio:

- Un documento potrebbe essere trasformato in un vettore di valori tf-idf.
- Un'immagine potrebbe essere trasformata in un vettore di caratteristiche estratte da una rete neurale (es. [1.0, 3.4, -1.2, 0.3, 1.4, ...]).
- Nell'esempio della slide, il risultato è un vettore come `[0.3, -1.2, 0.3, 1.4, ..., -0.03, 1.47]`.

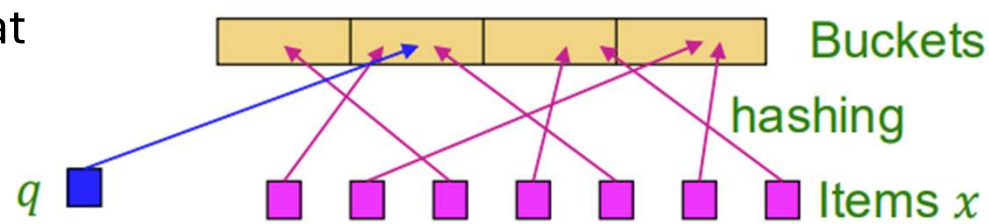


Una query viene inviata per trovare elementi simili a un dato oggetto (ad esempio, "trova documenti simili a questo").

Questo contenuto proviene da un'applicazione che ha bisogno di trovare elementi simili, ad esempio un motore di ricerca o un sistema di raccomandazione.

https://www.pinecone.io/learn/vector-database/

- LSH is really a family of related techniques
- In general, one throws items into buckets using several different “hash functions”.
- You examine only those pairs of items that share a bucket for at least one of these hashings.



Nel problema di **finding similar items**, dobbiamo confrontare un grande numero di oggetti (ad esempio documenti, utenti, immagini, prodotti) per identificare quelli che sono simili tra loro, secondo una certa misura di similarità (come la similarità di Jaccard, la similarità coseno o la distanza Euclidea). Tuttavia, quando lavoriamo con big data, il numero di oggetti può essere nell'ordine di milioni o miliardi, e ogni oggetto può essere rappresentato in uno spazio ad alta dimensionalità (ad esempio, un documento rappresentato come un vettore di parole con migliaia di dimensioni). Confrontare ogni coppia di oggetti in modo diretto richiederebbe un numero di confronti quadratico (
`O(n^2)`
), il che è computazionalmente impraticabile. È qui che entrano in gioco tecniche come il **vector space embedding** e l'**LSH**, che rendono il processo scalabile ed efficiente.
.

VECTOR SPACE EMBEDDING = La trasformazione degli oggetti in vettori numerici in uno spazio vettoriale, dove la similarità tra oggetti può essere misurata come la vicinanza tra vettori (ad esempio, usando la distanza Euclidea o la similarità coseno).

LSH = Una tecnica che permette di raggruppare oggetti simili in "bucket" (contenitori) usando funzioni di hash, riducendo il numero di confronti necessari. FAMIGLIA DI METODI

IDEA
Usare più funzioni hash per assegnare oggetti a bucket.

Ogni funzione di hash rappresenta una "proiezione" diversa dei dati, aumentando la probabilità che oggetti simili finiscano nello stesso bucket.

Una volta che gli oggetti sono stati raggruppati nei bucket, il sistema confronta solo le coppie di oggetti che condividono almeno un bucket. Questo è il "trucco" che rende l'LSH così efficiente: invece di fare
`O(n^2)`
confronti, si concentra solo su un piccolo sottoinsieme di coppie candidate.

Vector Space Embedding $\longleftrightarrow$ LSH

Our problem

- **Given: High dimensional data points $x_1, x_2, \ldots$**
  - **For example:** Image is a long vector of pixel colors

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} \rightarrow [1\ 2\ 1\ 0\ 2\ 1\ 0\ 1\ 0]$$

- **And some distance function $d(x_1, x_2)$**
  - Which quantifies the "distance" between x_1 and x_2

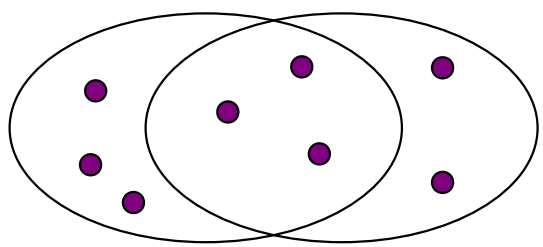
- **Goal:** Find **all pairs of data points** (x_i, x_j) that are within some distance threshold $d(x_i, x_j) \leq s$
- **Note:** Naïve solution would take $O(N^2)$ ☹

  where N is the number of data points

- **MAGIC: This can be done in $O(N)$!! How?**

Distance Measures

- **Goal: Find near-neighbors in high-dim. space**
  - We formally define "near neighbors" as points that are a "small distance" apart
- For each application, we first need to define what "**distance**" means
- **Example: Jaccard distance/similarity**
  - The **Jaccard similarity** of two **sets** is the size of their intersection divided by the size of their union: $\boldsymbol{sim(C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|}$
  - **Jaccard distance:** $\boldsymbol{d(C_1, C_2) = 1 - |C_1 \cap C_2| / |C_1 \cup C_2|}$



3 in intersection
8 in union
Jaccard similarity= 3/8
Jaccard distance = 5/8

Task: Finding Similar Documents

- **Goal: Given a large number (N in the millions or billions) of documents, find "near duplicate" pairs**
- **Applications:**
  - Plagiarism
  - Mirror websites, or approximate mirrors (Don't want to show both in search results)
  - Similar news articles at many news sites (Cluster articles by "same story")
- **Problems:**
  - Many small pieces of one document can appear out of order in another
  - Too many documents to compare all pairs
  - Documents are so large or so many that they cannot fit in main memory

3 Essential Steps for Similar Docs

1. ***Shingling:*** Convert documents to sets
2. ***Min-Hashing:*** Convert large sets to short signatures, while preserving similarity
3. ***Locality-Sensitive Hashing:*** Focus on pairs of signatures likely to be from similar documents
   - **Candidate pairs!**

Finding Similar Items nei Big Data

Introduzione

Ecco una spiegazione ampliata e più dettagliata dei contenuti delle slide:

1 Il Problema della Ricerca di Vicini in Dati ad Alta Dimensione

Il nostro obiettivo è lavorare con dati ad alta dimensione e identificare coppie di punti che siano vicini tra loro secondo una certa misura di distanza.

1.1 Definizione del Problema

Abbiamo un insieme di punti dati in uno spazio ad alta dimensione. Un esempio concreto potrebbe essere rappresentato da immagini digitali, che possono essere viste come vettori molto lunghi di valori corrispondenti ai colori dei pixel. Ad esempio, un'immagine può essere convertita in un vettore numerico come il seguente:

$$[1, 2, 1, 0, 2, 1, 0, 1, 0]$$

Per determinare quanto due punti dati siano vicini tra loro, dobbiamo definire una funzione di distanza $d(x_i, x_j)$ che quantifichi la distanza tra i punti x_i e x_j.

1.2 Obiettivo della Ricerca

Vogliamo trovare tutte le coppie di punti (x_i, x_j) che si trovano entro una certa soglia di distanza T, ovvero:

$$d(x_i, x_j) \leq T$$

Il problema principale è che un approccio ingenuo (confrontare ogni punto con tutti gli altri) richiederebbe un costo computazionale pari a $O(n^2)$, dove n è il numero totale di punti. Questo diventa insostenibile se i dati sono molto numerosi. Tuttavia, con tecniche più avanzate, è possibile ottenere una soluzione più efficiente con un costo computazionale molto inferiore, pari a $O(n)$.

2 Misure di Distanza

Per determinare se due punti sono vicini, è fondamentale definire un criterio per misurare la loro somiglianza o dissimilarità. Diverse applicazioni richiedono metriche diverse. Un esempio comune è la distanza di Jaccard, utilizzata per confrontare insiemi.

2.1 Definizione di Similarità e Distanza di Jaccard

Data una coppia di insiemi C_1 e C_2, la similarità di Jaccard è definita come il rapporto tra la dimensione dell'intersezione e la dimensione dell'unione dei due insiemi:

$$\text{sim}(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$$

Mentre la distanza di Jaccard è:

$$d(C_1, C_2) = 1 - \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$$

Ad esempio, se due insiemi hanno 3 elementi in comune e la loro unione contiene 8 elementi, otteniamo:

$$\text{sim}(C_1, C_2) = \frac{3}{8} = 0.375 \quad \text{e} \quad d(C_1, C_2) = 1 - 0.375 = 0.625$$

Questa misura è molto utile per problemi di confronto di documenti, dove vogliamo individuare testi simili o duplicati.

3 Problema della Ricerca di Documenti Simili

In molti scenari reali, dobbiamo confrontare un numero enorme di documenti (milioni o miliardi) e identificare coppie che siano simili.

3.1 Applicazioni Pratiche

- **Plagio**: individuare testi che siano stati copiati (parzialmente o completamente).
- **Riconoscimento di siti mirror**: evitare che motori di ricerca mostrino più copie dello stesso contenuto.
- **Raggruppamento di notizie**: identificare articoli che parlano dello stesso argomento su più fonti diverse.

3.2 Sfide Principali

- Parti di un documento possono apparire in ordine diverso in un altro documento.
- Confrontare ogni documento con tutti gli altri è computazionalmente troppo costoso.
- I documenti sono troppo numerosi o troppo grandi per essere gestiti interamente in memoria.

4 Strategia in Tre Passaggi per Confrontare Documenti

Per affrontare il problema in modo efficiente, si utilizzano tre tecniche chiave:

4.1 Shingling: Convertire i Documenti in Insiemi

Il primo passo è trasformare un documento in un insieme di "shingle" o sottostringhe di lunghezza fissa (ad esempio, gruppi di 3 parole consecutive).

- Ad esempio, il testo "il gatto nero corre" con shingle di 3 parole diventa:

  ["il gatto nero", "gatto nero corre"]

- Questo permette di rappresentare un documento come un insieme di stringhe e di usare misure come la similarità di Jaccard.

4.2 Min-Hashing: Creare una Firma Compatta

Poiché gli insiemi ottenuti dallo shingling possono essere molto grandi, si utilizza il Min-Hashing per ridurre la dimensione dei dati.

- Questo metodo crea una firma compatta per ogni documento, riducendo drasticamente lo spazio richiesto ma mantenendo la similarità tra documenti.
- L'idea è trasformare un insieme di shingle in una firma numerica più piccola, che può essere confrontata velocemente.

4.3 Locality-Sensitive Hashing (LSH): Identificare le Coppie Candidati

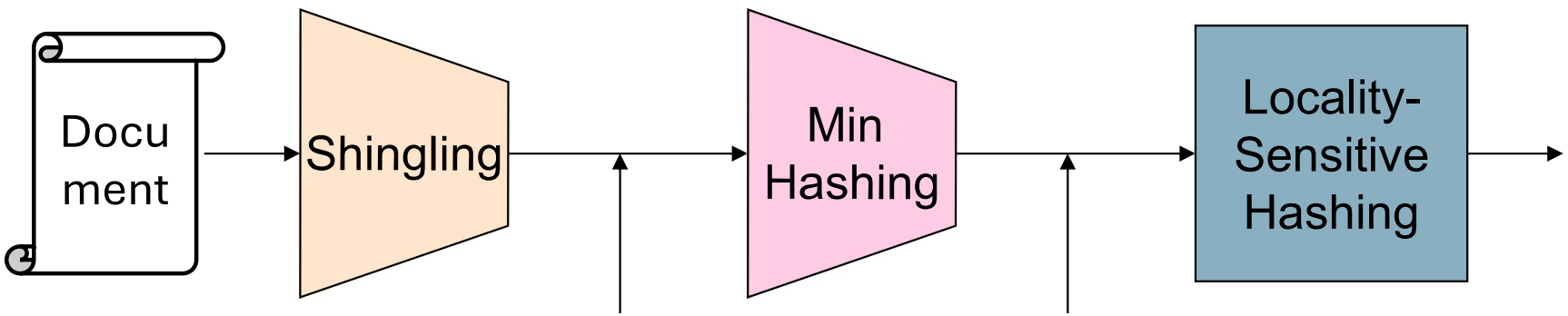
L'ultimo passo è usare Locality-Sensitive Hashing (LSH) per confrontare solo le firme che hanno alta probabilità di somiglianza.

- LSH raggruppa le firme in bucket in modo che solo i documenti probabilmente simili vengano confrontati tra loro.
- Questo riduce enormemente il numero di confronti necessari, migliorando le prestazioni.

5 Conclusione

Questa strategia consente di individuare documenti simili in modo efficiente, anche su dataset enormi, evitando il confronto di tutte le coppie possibili. L'uso di misure di distanza appropriate e tecniche di hashing consente di trovare rapidamente contenuti duplicati o correlati, con applicazioni che spaziano dalla ricerca web alla prevenzione del plagio.

The Big Picture



Candidate pairs: those pairs of signatures that we need to test for similarity

The set of strings of length ***k*** that appear in the doc-ument

Signatures: short integer vectors that represent the sets, and reflect their similarity

Rilevamento di Documenti Simili

L'immagine che hai condiviso rappresenta un processo utilizzato per trovare documenti simili in un insieme di dati, un problema noto come "finding similar items" o "near-duplicate detection". Questo processo è spesso utilizzato in contesti come la ricerca di documenti duplicati sul web, la raccomandazione di contenuti o il rilevamento di plagio. Di seguito spieghiamo passo per passo il concetto illustrato, basato su tecniche come shingling, min-hashing e locality-sensitive hashing (LSH).

1 Documento (Input)

Il processo inizia con un documento, che può essere un testo, una pagina web o qualsiasi altro tipo di contenuto. L'obiettivo è confrontare questo documento con altri per determinare se ci sono documenti simili.

2 Shingling

Il primo passo è il *shingling*. Questo processo trasforma il documento in un insieme di *shingle* (sottosequenze di lunghezza k). Uno shingle è una sequenza di k elementi consecutivi (ad esempio, k parole o k caratteri) estratti dal documento.

Esempio: Supponiamo che il documento sia "il gatto nero" e che $k = 2$ (bigramma di parole). Gli shingle saranno:

- "il gatto"
- "gatto nero"

Questo passaggio è importante perché due documenti simili avranno molti shingle in comune, mentre documenti diversi ne avranno pochi.

3 Min-Hashing

Il passo successivo è il *min-hashing*, che serve a creare una rappresentazione più compatta degli insiemi di shingle, chiamata *signature* (firma). Le signature sono vettori più corti che riflettono la somiglianza tra gli insiemi di shingle.

3.1 Come funziona il Min-Hashing

- Si applicano diverse funzioni di hash (ad esempio, $h_1, h_2, ..., h_n$) agli shingle del documento.
- Per ogni funzione di hash, si prende il valore minimo (da cui "min-hashing") tra tutti gli hash degli shingle di un documento.
- Questo processo genera un vettore di lunghezza n (dove n è il numero di funzioni di hash), che è la "signature" del documento.

Perché è utile? La signature conserva una proprietà importante: la probabilità che due documenti abbiano la stessa signature in una posizione è proporzionale alla loro somiglianza (misurata, ad esempio, con la similarità di Jaccard):

$$extsim(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|} \tag{1}$$

4 Locality-Sensitive Hashing (LSH)

Le signature generate dal min-hashing sono poi utilizzate con il *locality-sensitive hashing* (LSH) per identificare coppie di documenti che sono probabilmente simili, chiamate **candidate pairs**.

4.1 Come funziona LSH

- Le signature vengono divise in *bande* (sottovettori).
- Ogni banda viene inserita in una tabella di hash. Due documenti che hanno la stessa signature in una banda finiscono nello stesso *bucket* della tabella di hash.
- Le coppie di documenti che condividono almeno un bucket sono considerate **candidate pairs**, cioè coppie che vale la pena confrontare per verificare la loro effettiva somiglianza.

Obiettivo di LSH: LSH riduce drasticamente il numero di confronti necessari. Senza LSH, il confronto tra tutti i documenti avrebbe un costo $O(n^2)$. Con LSH, si identificano solo le coppie che hanno una buona probabilità di essere simili, riducendo il numero di confronti a un valore molto inferiore.

5 Candidate Pairs

Alla fine del processo, otteniamo le *candidate pairs*, ovvero le coppie di documenti che sono state identificate come potenzialmente simili. A questo punto, si applica un confronto più dettagliato (ad esempio, calcolando la similarità di Jaccard esatta tra i loro insiemi di shingle) per confermare se sono effettivamente simili.

6 Riassunto del Processo

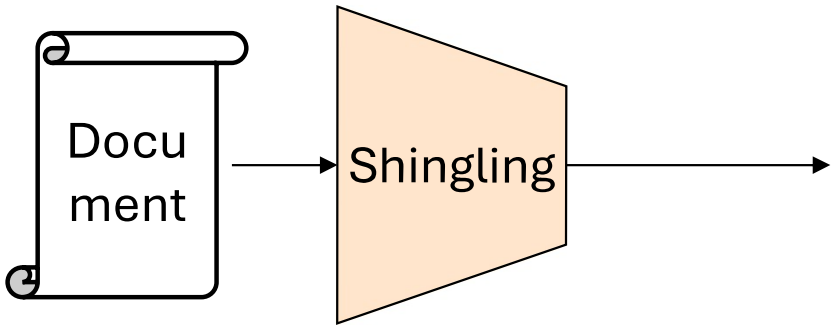
1. **Shingling**: Trasforma il documento in un insieme di shingle di lunghezza k.
2. **Min-Hashing**: Genera una signature (un vettore compatto) che rappresenta il documento e conserva la sua somiglianza con altri documenti.
3. **Locality-Sensitive Hashing (LSH)**: Identifica coppie di documenti che sono probabilmente simili (candidate pairs), riducendo il numero di confronti necessari.
4. **Confronto finale**: Le candidate pairs vengono analizzate più in dettaglio per confermare la loro somiglianza.

7 Esempio Pratico

Immagina di avere un database di 1 milione di pagine web e di voler trovare pagine quasi identiche (ad esempio, per rimuovere duplicati). Confrontare ogni pagina con ogni altra richiederebbe 500 miliardi di confronti! Usando questo processo:

- Trasformi ogni pagina in un insieme di shingle (ad esempio, sequenze di 3 parole).
- Generi una signature per ogni pagina con min-hashing (diciamo un vettore di 100 numeri).
- Usi LSH per trovare candidate pairs, riducendo il numero di confronti a poche migliaia.
- Infine, confronti solo le candidate pairs per confermare la loro somiglianza.

Questo approccio è molto più efficiente e scalabile, ed è ampiamente utilizzato in applicazioni reali come i motori di ricerca, i sistemi di raccomandazione e il rilevamento di contenuti duplicati.



Shingling

Convert documents to sets

Documents as High-Dim Data

- **Step 1: *Shingling:* Convert documents to sets**
- **Simple approaches:**
  - Document = set of words appearing in document
  - Document = set of "important" words
  - Don't work well for this application. Why?
- **Need to account for ordering of words!**
- A different way: **Shingles!**

Define: Shingles

- A *k*-shingle (or *k*-gram) for a document is a sequence of *k* tokens that appears in the doc
  - Tokens can be characters, words or something else, depending on the application
  - Assume tokens = characters for examples
- **Example:**
  - ❖ **k=2**
  - ❖ document $\mathbf{D_1}$ = abcab
  - ➢ Set of 2-shingles: $\mathbf{S(D_1)}$ = {ab, bc, ca}
    - **Option:** Shingles as a bag (multiset), count ab twice: $\mathbf{S'(D_1)}$ **=** {ab, bc, ca, ab}

Working Assumption

- **Documents that have lots of shingles in common have similar text, even if the text appears in different order**
- **Caveat:** You must pick ***k*** large enough, or most documents will have most shingles
  - ***k*** = 5 is OK for short documents
  - ***k*** = 10 is better for long documents

How large K should be?

- It depends on how long typical documents are and how large the set of typical character is.
  - Idea: k should be picked large enough that the probability of any given shingle appearing in any given document is low.
- **Example:**
  - ❖e-mails: only letters and white-space character
  - ➢With **k**=5 there would be $27^5 = 14M$ shingles
    - **Note:** an email usually is much smaller than 14M chars.

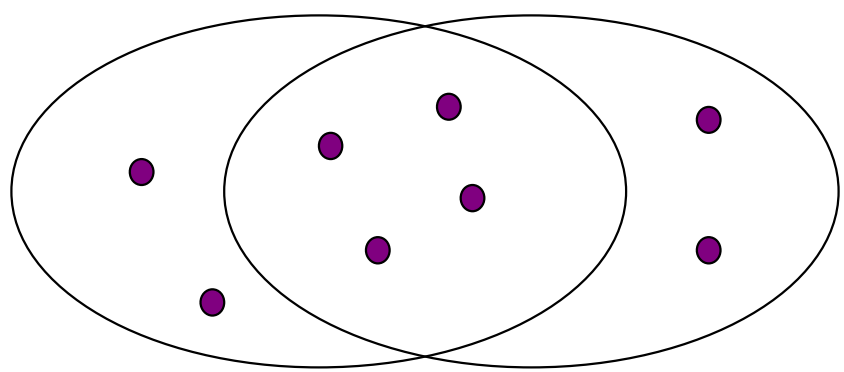
Compressing Shingles

- To **compress long shingles**, we can **hash** them to (say) 4 bytes
- **Represent a document by the set of hash values of its *k*-shingles**
- **Example:**
  - ❖ **k=2**; document $\mathbf{D_1}$= abcab
  - ➢ Set of 2-shingles: $\mathbf{S(D_1)}$ = {ab, bc, ca}
  - ➢ Hash the singles: $\mathbf{h(D_1)}$ = {1, 5, 7}

Similarity Metric for Shingles

- **Document D_1 is a set of its k-shingles $C_1=S(D_1)$**
  - Equivalently, each document is a 0/1 vector in the space of *k*-shingles
    - Each unique shingle is a dimension
    - Vectors are very sparse
- **A natural similarity measure is the Jaccard similarity:**

$$sim(D_1, D_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$$



Motivation for Minhash/LSH

- **Suppose we need to find near-duplicate documents among $N = 1$ million documents**
- Naïvely, we would have to compute **pairwise Jaccard similarities** for **every pair of docs**
  - $\boldsymbol{N(N-1)/2}$ **≈ 5*10^{11}** comparisons
  - At 10^5 secs/day and 10^6 comparisons/sec, it would take **5 days**
- For $\boldsymbol{N = 10}$ million, it takes more than a year…

Shingling: Convertire Documenti in Insiemi

1 Introduzione a Shingling

Il processo di shingling è fondamentale per trattare i documenti come dati ad alta dimensionalità, ossia rappresentarli in uno spazio vettoriale in cui ogni dimensione corrisponde a un elemento specifico del documento. L'obiettivo principale è trasformare il testo in una forma che permetta di calcolare la similarità tra documenti.

1.1 Approcci Semplici

Alcuni approcci semplici per convertire documenti in insiemi includono:

- **Documento = insieme di parole presenti nel documento**: Il documento viene ridotto a un insieme di parole uniche.
- **Documento = insieme di parole "importanti"**: Si selezionano solo le parole ritenute significative, eliminando stop words come "il", "e", "di".

Questi metodi non funzionano bene perché ignorano l'ordine delle parole, che è cruciale per comprendere il significato del testo. Per catturare l'ordine delle parole, si usa il concetto di **shingles**.

2 Definizione di Shingles

Uno **shingle** (o **k-gram**) è una sequenza contigua di k token che appare in un documento. I token possono essere caratteri, parole o altre unità linguistiche, a seconda dell'applicazione.

2.1 Esempio con $k = 2$

Consideriamo il documento $D_1 =$ "abcab":

- Sequenze di 2 caratteri (shingles): $S(D_1) = \{\text{"ab"}, \text{"bc"}, \text{"ca"}\}$.
- Se consideriamo gli shingles come un **multiset** (insieme con ripetizioni), potremmo contare più volte gli shingles duplicati: $S'(D_1) = \{\text{"ab"}, \text{"bc"}, \text{"ca"}, \text{"ab"}\}$.

3 Assunzione di Lavoro

L'assunzione di base è che **documenti con molti shingles in comune hanno testi simili**, anche se le parole appaiono in ordine diverso. Tuttavia, è importante scegliere un valore di k adeguato:

- $k = 5$ è adatto per documenti brevi (es., tweet, SMS).
- $k = 10$ è più appropriato per documenti lunghi (es., articoli, report).

4 Dimensione di k: Quanto Grande Deve Essere?

La scelta di k dipende dalla lunghezza tipica dei documenti e dal vocabolario dei caratteri/token utilizzati.

4.1 Esempio: E-mail

- Caratteri tipici: lettere (26) + spazi bianchi (1) = 27 caratteri totali.
- Con $k = 5$, il numero totale di possibili shingles è $27^5 = 14.348.907$.
- Una singola e-mail ha solitamente meno di 14 milioni di caratteri, quindi $k = 5$ è sufficientemente grande per garantire che la probabilità di un particolare shingle di apparire in un documento sia bassa.

5 Compressione degli Shingles

Gli shingles possono essere molto lunghi, soprattutto per valori grandi di k. Per ridurre lo spazio necessario, si può applicare una funzione hash per comprimerli. Ad esempio:

- Hashare ogni shingle in un numero intero di 4 byte (32 bit).

5.1 Esempio di Compressione

Consideriamo $k = 2$, Documento $D_1 =$ "abcab":

- Shingles: $S(D_1) = \{\text{"ab"}, \text{"bc"}, \text{"ca"}\}$.
- Applichiamo una funzione hash: $h(\text{"ab"}) = 1$, $h(\text{"bc"}) = 5$, $h(\text{"ca"}) = 7$.
- Rappresentazione compressa: $h(D_1) = \{1, 5, 7\}$.

6 Metrica di Similarità per Shingles

Ogni documento può essere rappresentato come un insieme di shingles. Equivalentemente, ogni documento può essere visto come un vettore binario nello spazio degli shingles, dove ogni dimensione corrisponde a uno shingle unico.

6.1 Caratteristiche dei Vettori

- Sono molto sparsi, poiché la maggior parte degli shingles possibili non appare in un dato documento.

Una metrica naturale per misurare la similarità tra due documenti è la **similarità di Jaccard**:

$$\text{sim}(D_1, D_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$$

dove:

- C_1 e C_2 sono gli insiemi di shingles dei documenti D_1 e D_2,
- $|C_1 \cap C_2|$ è il numero di shingles comuni,
- $|C_1 \cup C_2|$ è il numero totale di shingles distinti nei due documenti.

7 Motivazione per Minhash/LSH

Calcolare la similarità di Jaccard per ogni coppia di documenti su larga scala è computazionalmente proibitivo. Ad esempio:

- Con $N = 1$ milione di documenti, il numero di confronti necessari è:

$$\frac{N(N-1)}{2} \approx 5 \times 10^{11}.$$

- Supponendo di poter effettuare 10^6 confronti al secondo, ci vorrebbero circa 5 giorni.

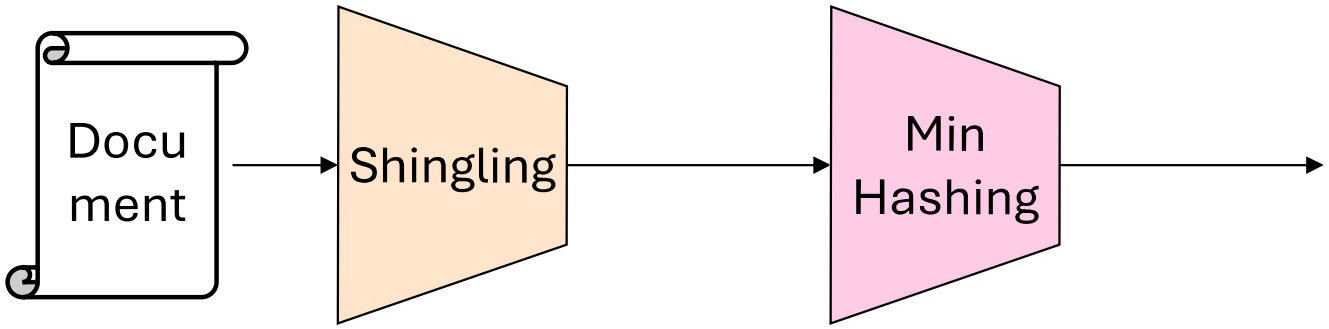
7.1 Soluzione: Minhash e Locality-Sensitive Hashing (LSH)

- **Minhash**: Riduce la dimensione dei vettori mantenendo la similarità di Jaccard.
- **LSH**: Consente di identificare rapidamente coppie di documenti simili senza dover confrontare tutte le coppie.

Con queste tecniche, è possibile scalare l'analisi della similarità a milioni o miliardi di documenti.

8 Conclusione

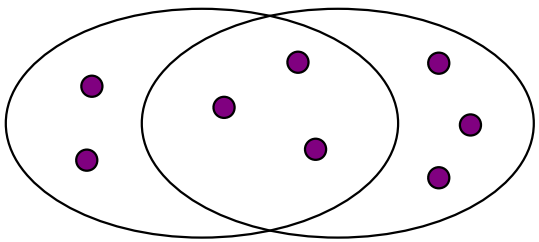
Il processo di shingling è uno strumento potente per rappresentare documenti come insiemi di elementi ordinati (shingles). Questa rappresentazione cattura l'ordine delle parole e permette di calcolare la similarità tra documenti utilizzando metriche come la similarità di Jaccard. Tuttavia, per gestire grandi quantità di dati, è necessario combinare shingling con tecniche di hashing efficienti come Minhash e LSH.



Convert large sets to short signatures, while preserving similarity

Min-Hashing

Encoding Sets as Bit Vectors



- Many similarity problems can be formalized as **finding subsets that have significant intersection**
- **Encode sets using 0/1 (bit, boolean) vectors**
  - One dimension per element in the universal set
- Interpret set intersection as bitwise **AND**, and set union as bitwise **OR**

- **Example:** $\mathbf{C_1}$ = 10111; $\mathbf{C_2}$ = 10011
  - Size of intersection **= 3**; size of union **= 4**,
  - **Jaccard similarity** (not distance) **= 3/4**
  - **Distance: $d(C_1,C_2)$ = 1 – (Jaccard similarity) = 1/4**

From Sets to Boolean Matrices

$M[e,s] \rightarrow e \in S?$ 1 si / 0 no

Colonne = doc
riga - elementi

- **Rows** = elements (shingles)
- **Columns** = sets (documents)
  - $M[e, s] = 1$ if and only if ***e*** is a member of ***s***
  - Column similarity is the Jaccard similarity of the corresponding sets (rows with value *1)*
  - **Typical matrix is sparse!**
- **Each document is a column:**
  - **Example: sim(C_1 ,C_2) = ?**
    - Size of intersection = 3
    - Size of union = 6
    - Jaccard similarity (not distance) = 3/6
    - **d(C_1,C_2) = 1 – (Jaccard similarity) = 3/6**

Documents

Shingles

1	1	1	0
1	1	0	1
0	1	0	1
0	0	0	1
1	0	0	1
1	1	1	0
1	0	1	0

Outline: Finding Similar Columns

- **So far:**
  - Documents → Sets of shingles
  - Represent sets as boolean vectors in a matrix
- **Next goal: Find similar columns while computing small signatures**
  - **Similarity of columns == similarity of signatures**

Next goal → TROVARE colonne simili
CALCOLANDO firme di dimensioni ridotte
Assumendo similarità colonne == similarità firme

Outline: Finding Similar Columns

- **Next Goal: Find similar columns, Small signatures**
- **Naïve approach:**
  1. **Signatures of columns:** small summaries of columns
  2. **Examine pairs of signatures** to find similar columns
     - **Essential:** Similarities of signatures and columns are related
  3. **Optional:** Check that columns with similar signatures are really similar
- **Warnings:**
  - Comparing all pairs may take too much time: **Job for LSH**
    - These methods can produce false negatives, and even false positives (if the optional check is not made)

Next Goal = Il prossimo passo è trovare colonne simili, ma invece di confrontare direttamente le colonne (che possono essere molto grandi), si utilizzano **small signatures** (firme di dimensioni ridotte). Queste firme sono delle rappresentazioni compatte di ogni colonna, calcolate in modo tale da preservare informazioni sulla similarità. Un esempio tipico di come ottenere queste firme è il **Min-Hashing**, dove si applicano funzioni di hash per generare un vettore di valori minimi che riassumono la colonna.

Naive Approach = L'approccio più semplice per trovare colonne simili usando le firme può essere scomposto in tre punti:

1. **Signatures of columns: small summaries of columns**
   Per ogni colonna (documento), si calcola una firma, che è un riassunto compatto. Ad esempio, con il Min-Hashing, la firma è un vettore di (k) valori, dove ogni valore è il minimo hash degli shingles presenti nella colonna, calcolato con una funzione di hash diversa. Queste firme sono molto più piccole della matrice originale, il che le rende efficienti da manipolare.
2. **Examine pairs of signatures to find similar columns**
   Si confrontano tutte le coppie di firme per determinare quali siano simili. La similarità tra firme può essere misurata, ad esempio, calcolando la frazione di posizioni in cui coincidono (approssimando il coefficiente di Jaccard). Se due firme sono simili, si presume che le colonne corrispondenti (i documenti) siano simili.
3. **Optional: Check that columns with similar signatures are really similar**
   Questo passo è facoltativo ma importante. Anche se le firme sono simili, non garantiscono al 100% che le colonne lo siano, a causa della natura approssimata del metodo. Quindi, si può verificare direttamente la similarità tra le colonne originali (ad esempio calcolando il coefficiente di Jaccard sugli shingles) per confermare i risultati. Questo riduce i falsi positivi.

⚠ Ci sono alcune sfide e limitazioni da considerare:

- **Comparing all pairs may take too much time: Job for LSH**
  Confrontare tutte le possibili coppie di firme (che possono essere `O(n^2)` per (n) colonne) diventa impraticabile quando il numero di documenti è molto grande. Qui entra in gioco il **Locality-Sensitive Hashing (LSH)**, una tecnica che accelera il processo. LSH raggruppa colonne che hanno probabilità più alta di essere simili in base alle loro firme, riducendo il numero di confronti necessari. Invece di verificare ogni coppia, LSH usa hash bucket per filtrare i candidati, concentrandosi solo su quelli potenzialmente simili.
- **These methods can produce false negatives, and even false positives (if the optional check is not made)**
  - **Falsi negativi**: Alcuni documenti simili potrebbero non essere identificati se le loro firme differiscono a causa della casualità delle funzioni di hash. Questo è un compromesso accettabile per guadagnare efficienza.
  - **Falsi positivi**: Se non si esegue il controllo opzionale (punto 3), due colonne con firme simili potrebbero non essere realmente simili. Questo succede perché le firme sono solo un'approssimazione. Il controllo opzionale mitiga questo problema confrontando direttamente le colonne quando necessario.

Hashing Columns (Signatures)

- **Key idea:** "hash" each column ***C*** to a small ***signature h(C)***, such that:
  1. ***h(C)*** is small enough that the signature fits in RAM
  2. ***sim(C_1, C_2)*** is the same as the "similarity" of signatures ***h(C_1)*** and ***h(C_2)***

- **Goal: Find a hash function *h(·)* such that:**
  - If ***sim(C_1,C_2)*** is high, then with high prob. ***h(C_1) = h(C_2)***
  - If ***sim(C_1,C_2)*** is low, then with high prob. ***h(C_1) ≠ h(C_2)***

- **Hash docs into buckets. Expect that "most" pairs of near duplicate docs hash into the same bucket!**

Hashing delle Colonne per Creare Firme nel Contesto del Rilevamento di Documenti Simili

Hashing Columns (Signatures)

L'idea centrale è trasformare ogni colonna C di una matrice (dove ogni colonna rappresenta un documento e ogni riga uno shingle) in una firma più piccola, chiamata $h(C)$, usando una funzione di hash. Questo processo serve a rendere più efficiente il confronto tra documenti, mantenendo però una relazione con la similarità originale tra le colonne.

Proprietà Fondamentali della Firma $h(C)$

Ci sono due proprietà fondamentali che la firma $h(C)$ deve soddisfare:

1. $h(C)$ **deve essere sufficientemente piccola da poter essere caricata in memoria (RAM)**.
   La firma $h(C)$ è un vettore molto più corto rispetto alla colonna originale, spesso composto da poche decine o centinaia di valori interi. Nel contesto del Min-Hashing, la firma di una colonna è un vettore di k valori, dove k è il numero di funzioni di hash utilizzate. Ad esempio, se $k = 100$, la firma è un vettore di 100 numeri interi, che è molto più gestibile rispetto a un vettore con milioni di bit.

2. **La similarità tra due colonne C_1 e C_2 deve essere approssimativamente uguale alla similarità tra le loro firme $h(C_1)$ e $h(C_2)$.**
   In altre parole, la funzione di hash deve preservare la similarità: se C_1 e C_2 sono simili, allora anche $h(C_1)$ e $h(C_2)$ dovrebbero essere simili, e viceversa. Nel Min-Hashing, questa proprietà è garantita in termini probabilistici: la probabilità che $h(C_1) = h(C_2)$ in una data posizione della firma è esattamente uguale al coefficiente di Jaccard tra C_1 e C_2. Confrontando le firme (ad esempio, calcolando la frazione di posizioni in cui $h(C_1)$ e $h(C_2)$ coincidono), si ottiene una stima della similarità originale.

Obiettivo

L'obiettivo è trovare una funzione di hash $h(\cdot)$ che soddisfi due proprietà probabilistiche, in modo da riflettere la similarità tra colonne nella similarità tra firme:

- Se $\text{sim}(C_1, C_2)$ è alta, allora con alta probabilità $h(C_1) = h(C_2)$. Questo assicura che i documenti simili vengano riconosciuti come tali.

- Se $\text{sim}(C_1, C_2)$ è bassa, allora con alta probabilità $h(C_1) \neq h(C_2)$. Questo riduce i falsi positivi, evitando di identificare come simili documenti che non lo sono.

Queste due proprietà descrivono una caratteristica fondamentale di una funzione di hash locality-sensitive (*sensibile alla località*). Una funzione di hash con queste proprietà garantisce che coppie di colonne simili tendano a produrre firme simili, mentre coppie di colonne diverse producano firme diverse. Il Min-Hashing è un esempio di una tecnica che soddisfa queste proprietà, perché la probabilità che due colonne abbiano lo stesso valore minimo di hash è proporzionale alla loro similarità di Jaccard.

Hashing in Bucket

Una volta calcolate le firme $h(C)$, queste vengono utilizzate per raggruppare i documenti in bucket (*contenitori*) usando una tecnica come il Locality-Sensitive Hashing (LSH). L'idea è la seguente:

- Si applica un'ulteriore funzione di hash (o si usano le firme stesse) per mappare ogni firma $h(C)$ in un bucket. Documenti con firme simili (o identiche) hanno un'alta probabilità di essere mappati nello stesso bucket.
- Ci si aspetta che "la maggior parte" delle coppie di documenti quasi identici (*near duplicates*) finiscano nello stesso bucket. Questo è il cuore dell'LSH: invece di confrontare tutte le possibili coppie di documenti (che richiederebbe $O(n^2)$ confronti per n documenti), ci si concentra solo sui documenti che cadono nello stesso bucket, riducendo drasticamente il numero di confronti necessari.

Ad esempio, nel contesto del Min-Hashing con LSH, si divide la firma in bande (*sotto-vettori*) e si applica un hash a ogni banda. Se due firme coincidono in almeno una banda, i documenti corrispondenti sono considerati "candidati" per essere simili e vengono confrontati più approfonditamente. Questo approccio aumenta l'efficienza, anche se può introdurre falsi negativi (documenti simili che non finiscono nello stesso bucket) o falsi positivi (documenti non simili che finiscono nello stesso bucket).

Collegamento con il Min-Hashing

Per contestualizzare, vediamo come il Min-Hashing si inserisce in questa slide:

- **Calcolo della firma** $h(C)$: Nel Min-Hashing, si generano k funzioni di hash (ad esempio, permutazioni casuali degli shingles). Per ogni colonna C, si calcola il valore minimo di hash per gli shingles presenti in C, per ciascuna delle k funzioni. La firma $h(C)$ è quindi un vettore di k valori interi.
- **Relazione tra similarità**: La probabilità che $h(C_1) = h(C_2)$ in una data posizione della firma è uguale al coefficiente di Jaccard tra C_1 e C_2. Questo soddisfa il requisito che la similarità delle firme riflette la similarità delle colonne.

Esempio Pratico

Supponiamo di avere due documenti D_1 e D_2, rappresentati come insiemi di shingles:

$$D_1 : \{\text{shingle1, shingle2, shingle3}\}, \quad D_2 : \{\text{shingle2, shingle3, shingle4}\}.$$

La matrice di incidenza avrà una colonna per ogni documento e una riga per ogni shingle. Supponiamo di avere 5 shingles totali (shingle1, shingle2, shingle3, shingle4, shingle5). La matrice sarà:

Shingle	D_1	D_2
shingle1	1	0
shingle2	1	1
shingle3	1	1
shingle4	0	1
shingle5	0	0

Coefficiente di Jaccard:

$$\text{sim}(D_1, D_2) = \frac{|D_1 \cap D_2|}{|D_1 \cup D_2|} = \frac{2}{4} = 0.5.$$

Con il Min-Hashing:

- Applichiamo $k = 2$ funzioni di hash casuali h_1 e h_2. Supponiamo che h_1 e h_2 producano valori interi per ogni shingle.
- Per D_1, calcoliamo il valore minimo di h_1 e h_2 tra gli shingles presenti (shingle1, shingle2, shingle3).
- Per D_2, facciamo lo stesso con (shingle2, shingle3, shingle4).
- Le firme $h(D_1)$ e $h(D_2)$ saranno due vettori di lunghezza 2. Se $\text{sim}(D_1, D_2) = 0.5$, ci aspettiamo che circa il 50% delle posizioni delle firme coincidano (probabilisticamente).

Poi, queste firme possono essere usate con LSH per raggruppare D_1 e D_2 nello stesso bucket, se sono abbastanza simili.

Conclusione

Questa slide descrive il cuore del processo di hashing per trovare documenti simili:

- Si creano firme piccole ($h(C)$) per ogni colonna.
- La funzione di hash è progettata per preservare la similarità (alta similarità $\rightarrow$ firme simili, bassa similarità $\rightarrow$ firme diverse).
- Le firme vengono usate per raggruppare documenti in bucket, aspettandosi che i documenti quasi identici finiscano nello stesso bucket.

Il Min-Hashing, combinato con LSH, è un'implementazione pratica di questa idea, che bilancia efficienza e accuratezza nel rilevamento di similarità.

Min-Hashing

- **Goal: Find a hash function *h(·)* such that:**
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$

- **Clearly, the hash function depends on the similarity metric:**
  - Not all similarity metrics have a suitable hash function

- **There is a suitable hash function for the Jaccard similarity:** It is called **Min-Hashing**

Min-Hashing

- Imagine the rows of the boolean matrix permuted under **random permutation π**

- Define a **"hash" function $h_\pi(C)$** = the index of the **first** (in the permuted order π) row in which column ***C*** has value **1**:

$$h_\pi(C) = min_\pi \pi(C)$$

- Use several (e.g., 100) independent hash functions (that is, permutations) to create a signature of a column

Min-Hashing: Una Tecnica per Approssimare la Similarità di Jaccard

1. Obiettivo Principale

Il **Min-Hashing** mira a trovare una funzione hash $h(\cdot)$ con le seguenti proprietà:

- Se la similarità tra due colonne C_1 e C_2 (misurata ad esempio con Jaccard) è alta, allora con alta probabilità $h(C_1) = h(C_2)$.
- Se la similarità è bassa, con alta probabilità $h(C_1) \neq h(C_2)$.

Questo permette di approssimare la similarità tra colonne usando firme compatte.

2. Dipendenza dalla Metrica di Similarità

La funzione hash dipende dalla metrica di similarità scelta. Non tutte le metriche (es. distanza euclidea) hanno una funzione hash adatta, ma il Min-Hashing è specificamente efficace per la **similarità di Jaccard**, definita come:

$$\text{Jaccard}(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$$

3. Meccanismo del Min-Hashing

Si immagina di permutare casualmente le righe di una matrice booleana (dove colonne = documenti, righe = shingles) con una permutazione π.

La funzione hash $h_\pi(C)$ è definita come l'indice della prima riga (nell'ordine permutato π) in cui la colonna C ha valore 1:

$$h_\pi(C) = \min_\pi \pi(C)$$

Si utilizzano diverse permutazioni indipendenti (es. 100) per creare una **firma (signature)** per ogni colonna, un vettore di valori che rappresenta il documento in modo compatto.

4. Conclusione

Il Min-Hashing genera firme che preservano la similarità di Jaccard, permettendo confronti efficienti tra grandi insiemi di dati. Questa tecnica è particolarmente utile in contesti in cui la scalabilità e l'efficienza computazionale sono cruciali, come nel rilevamento di duplicati, clustering e sistemi di raccomandazione.

Min-Hashing Example

2nd element of the permutation is the first to map to a 1

Permutation π

2)	2	4	3
3)	3	2	4
7)	7	1	7
6)	6	3	2
1)	1	6	6
5)	5	7	1
4)	4	5	5

Input matrix (Shingles x Documents)

1	0	1	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0

Signature matrix *M*

2	1	2	1
2	1	4	1
1	2	1	2

4th element of the permutation is the first to map to a 1

Data una matrice binaria in input, in cui le colonne rappresentano i documenti e le righe gli shingles, viene definito un insieme di k permutazioni scelto dal progettista. La matrice delle firme, ottenuta in output, avrà un numero di colonne pari al numero di documenti e un numero di righe pari al numero di permutazioni selezionate dal progettista. Per ogni permutazione, si considera il corrispondente vettore di permutazione e si identifica il primo valore al suo interno. Una volta trovato, si scorre la riga corrispondente nella matrice binaria fino a individuare il primo '1'; l'indice di colonna corrispondente a tale '1' viene quindi riportato nella matrice delle firme. Questo processo si ripete fino a completare la matrice delle firme.

The Min-Hash Property

0	0
0	0
1	**1**
0	0
0	1
1	0

- **Choose a random permutation π**
- **<u>Claim:</u> $\Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$**
- **Why?**
  - Let **X** be a doc (set of shingles), $y \in X$ is a shingle
  - **Then: $\Pr[\pi(y) = \min(\pi(X))] = 1/|X|$**
    - It is equally likely that any $y \in X$ is mapped to the ***min*** element
  - Let y be s.t. $\pi(y) = \min(\pi(C_1 \cup C_2))$
  - **Then either:** $\pi(y) = \min(\pi(C_1))$ if $y \in C_1$, **or**
    $\pi(y) = \min(\pi(C_2))$ if $y \in C_2$
  - So the prob. that **both** are true is the prob. $y \in C_1 \cap C_2$
  - **$\Pr[\min(\pi(C_1))=\min(\pi(C_2))]=|C_1 \cap C_2|/|C_1 \cup C_2| = sim(C_1, C_2)$**

One of the two cols had to have 1 at position y

Proprietà del Min-Hash

La slide spiega la proprietà del **Min-Hash**, una tecnica usata per stimare la similarità tra due insiemi (ad esempio, documenti rappresentati come insiemi di shingles). Ecco una spiegazione chiara e concisa.

Obiettivo

Stimare la similarità tra due insiemi C_1 e C_2 (ad esempio, due documenti). La similarità è definita come:

$$\text{sim}(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|},$$

ovvero la frazione di elementi comuni rispetto all'unione degli insiemi.

Procedura

1. Si sceglie una permutazione casuale π degli shingles (righe della matrice binaria).
2. Per ogni documento (colonna della matrice), si cerca il primo shingle (riga) che ha valore 1, seguendo l'ordine della permutazione π. Questo valore è il **Min-Hash** del documento:

$$h_\pi(C) = \min(\pi(C)).$$

Proprietà del Min-Hash

La probabilità che due documenti C_1 e C_2 abbiano lo stesso valore Min-Hash ($h_\pi(C_1) = h_\pi(C_2)$) è esattamente uguale alla loro similarità:

$$\Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2).$$

Perché funziona?

- Consideriamo un documento X come un insieme di shingles. La probabilità che uno shingle $y \in X$ sia il minimo dopo la permutazione è $\frac{1}{|X|}$, perché la permutazione è casuale.
- Se y è il minimo di $C_1 \cup C_2$, allora $h_\pi(C_1) = h_\pi(C_2)$ se e solo se $y \in C_1 \cap C_2$.
- La probabilità che $y \in C_1 \cap C_2$ è proprio:

$$\frac{|C_1 \cap C_2|}{|C_1 \cup C_2|},$$

  che è la similarità.

Esempio nella matrice

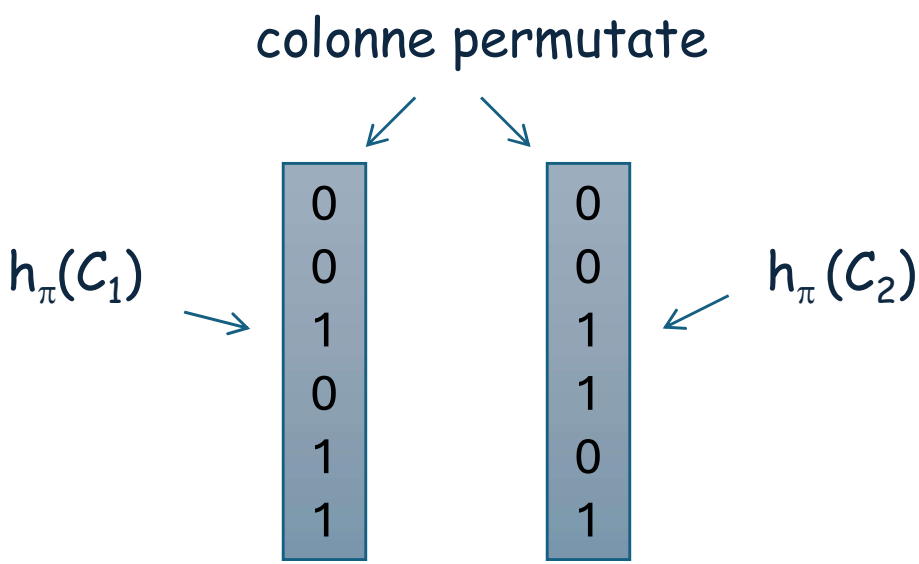
La matrice binaria mostra due documenti (colonne) e vari shingles (righe). Un "1" indica che lo shingle è presente nel documento. Applicando una permutazione π, si trova il primo "1" per ogni colonna, e si confrontano i risultati per stimare la similarità.

In sintesi

Il **Min-Hash** usa permutazioni casuali per mappare insiemi in valori rappresentativi, e la probabilità che due insiemi abbiano lo stesso valore Min-Hash riflette la loro similarità.

Analisi

- Qual è la probabilità che $h_l(C_1) = h_l(C_2)$?



$$\Pr ob[h_l(C_1) = h_l(C_2)] = \frac{\text{righe con entrambe le colonne 1}}{\text{righe con almeno una delle due colonne 1}} = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|} = Sim(C_1, C_2)$$

Analisi

Qual è la similarità tra $Sig(C_1)$ e $Sig(C_2)$?

$Sig(C_1)$	$Sig(C_2)$
$h_1(C_1)$	$h_1(C_2)$
$h_2(C_1)$	$h_2(C_2)$
…	…
…	…
$h_k(C_1)$	$h_k(C_2)$

$$Sim(Sign(C_1), Sign(C_2)) = \frac{\text{numero di righe in cui } h_l(C_1) = h_l(C_2)}{\text{numero totale di righe}}$$

$$Sim(Sign(C_1), Sign(C_2)) \longrightarrow \Pr ob[h_l(C_1) = h_l(C_2)] = Sim(C_1, C_2)$$

Domanda

Qual è la probabilità che $h_\pi(C_1) = h_\pi(C_2)$?

Contesto

Abbiamo due documenti C_1 e C_2 rappresentati come colonne in una matrice binaria (0 e 1 indicano la presenza/assenza di shingles). Usiamo una permutazione casuale π per calcolare i Min-Hash $h_\pi(C_1)$ e $h_\pi(C_2)$, ovvero il primo shingle (riga) con valore 1 seguendo l'ordine di π.

Formula

La probabilità che $h_\pi(C_1) = h_\pi(C_2)$ è uguale alla similarità tra C_1 e C_2, definita come:

$$\Pr[h_\pi(C_1) = h_\pi(C_2)] = \frac{\text{righe con entrambe le colonne a 1}}{\text{righe con almeno una delle due colonne a 1}} = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|} = \mathrm{Sim}(C_1, C_2).$$

Significato

Se C_1 e C_2 hanno molti shingles in comune, è più probabile che il loro Min-Hash sia lo stesso.

Domanda

Qual è la similarità tra $\mathrm{Sig}(C_1)$ e $\mathrm{Sig}(C_2)$?

Contesto

La firma di un documento ($\mathrm{Sig}(C_1)$, $\mathrm{Sig}(C_2)$) è un vettore di Min-Hash calcolato usando k permutazioni diverse $(h_1, h_2, \ldots, h_k)$. Ogni riga della firma contiene un valore Min-Hash per il documento.

Calcolo della similarità

La similarità tra le firme $\mathrm{Sig}(C_1)$ e $\mathrm{Sig}(C_2)$ è data dalla frazione di righe in cui i valori Min-Hash sono uguali:

$$\mathrm{Sim}(\mathrm{Sig}(C_1), \mathrm{Sig}(C_2)) = \frac{\text{numero di righe in cui } h_i(C_1) = h_i(C_2)}{\text{numero totale di righe}}.$$

Relazione con la similarità reale

In media, questa similarità approssima la similarità reale tra C_1 e C_2:

$$\mathrm{Sim}(\mathrm{Sig}(C_1), \mathrm{Sig}(C_2)) \to \Pr[h_\pi(C_1) = h_\pi(C_2)] = \mathrm{Sim}(C_1, C_2).$$

Significato

Usando più permutazioni (ossia più righe nella firma), la stima della similarità diventa più accurata.

Correttezza del Min-Hashing

- Definiamo $Sim^*(C_i,C_j) = Sim(Sig(C_i),Sig(C_j))$
- Sia s una soglia di similarità al di sopra della quale consideriamo due colonne "altamente simili". Supponiamo che s sia limitato inferiormente da una costante c.

- **Teorema:** Siano $0 < \delta < 1$, $\varepsilon > 0$, $k > 2\,\delta^{-2} c^{-1} \log \varepsilon^{-1}$. Allora per ogni coppia di colonne C_i , C_j valgono le seguenti proprietà:
  a. Se $Sim(C_i,C_j) \geq s \geq c$, allora $Prob[Sim^*(C_i,C_j) \geq (1-\delta)\, s] \geq 1 - \varepsilon$
  b. *Se* $Sim(C_i,C_j) \leq c$ allora $Prob[Sim^*(C_i,C_j) \leq (1+\delta)\, c] \geq 1 - \varepsilon$

- Il teorema ci dice che il numero di falsi positivi e negativi è limitato

Dimostrazione

- Dimostriamo la proprietà *a*. Per la *b* la dimostrazione è simile.
- Fissata una coppia di colonne C_i, C_j con $Sim(C_i,C_j) \geq s$ dobbiamo dimostrare che:

$$\text{Prob}[Sim^*(C_i, C_j) < (1-\delta)\cdot s] < \varepsilon$$

- Consideriamo la variabile aleatoria $X = X_1+\ldots+X_k$ dove X_l vale 1 se $h_l(C_i) = h_l(C_j)$ altrimenti vale 0.

- Il Chernoff Bound ci dice che:

$$\text{Prob}[X < (1-\delta)\cdot E[X]] < e^{-\frac{\delta^2 E[X]}{2}}$$

Dimostrazione

- Inoltre:
  - $Sim^*(C_i,C_j) = X / k$
  - $E[X] = k * Sim(C_i,C_j)$ (previsione di X)

- Quindi:

$$\mathrm{Prob}[Sim^*(C_i,C_j) < (1-\delta)\cdot s] = \mathrm{Prob}[\frac{X}{k} < (1-\delta)\cdot s] =$$

$$= \mathrm{Prob}[X < (1-\delta)\cdot k\cdot s] \leq \mathrm{Prob}[X < (1-\delta)\cdot E[X]] <$$

$$< e^{-\frac{\delta^2\cdot E[X]}{2}} = e^{-\frac{\delta^2\cdot k\cdot Sim(C_i,C_j)}{2}} \leq e^{-\frac{\delta^2\cdot k\cdot s}{2}} < \varepsilon$$

Similarity for Signatures

- We know: **$\Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$**
- Now generalize to multiple hash functions

- **The *similarity of two signatures* is the fraction of the hash functions in which they agree**

- **Note:** Because of the Min-Hash property, the similarity of columns is the same as the expected similarity of their signatures

Similarità tra Firme nel Contesto del Min-Hashing

La slide riguarda la **similarità tra firme** nel contesto del **Min-Hashing**, una tecnica usata per stimare la similarità tra insiemi di dati, come documenti o colonne in una matrice, in modo efficiente.

Principio di Base

Si parte dal fatto che, per una singola funzione hash casuale h_π, la probabilità che:

$$h_\pi(C_1) = h_\pi(C_2)$$

(dove C_1 e C_2 sono due colonne o insiemi) è uguale alla loro similarità:

$$\text{sim}(C_1, C_2).$$

Questo si basa sulla proprietà del **Min-Hash**, che associa il valore minimo di hash a un insieme.

Generalizzazione a Più Funzioni Hash

Estendendo il concetto a più funzioni hash, si costruiscono **firme (signature)** per C_1 e C_2 come vettori di valori minimi ottenuti da diverse funzioni hash. La similarità tra queste firme è definita come la frazione di funzioni hash in cui i valori coincidono:

$$\text{Similarità tra firme} = \frac{\text{numero di posizioni in cui le firme sono uguali}}{\text{totale delle funzioni hash}}.$$

Proprietà del Min-Hash

Grazie a questa proprietà, la similarità calcolata tra le firme approssima la similarità attesa tra le colonne originali. In altre parole, la frazione di accordo tra le firme riflette accuratamente la similarità intrinseca tra C_1 e C_2, rendendo il **Min-Hash** un metodo efficace per confronti su larga scala.

In Sintesi

Il **Min-Hashing** usa più funzioni hash per creare firme rappresentative, e la loro similarità, misurata come frazione di accordi, è un'estimazione affidabile della similarità reale tra i dati originali.

Min-Hashing Example

Permutation π

2	4	3
3	2	4
7	1	7
6	3	2
1	6	6
5	7	1
4	5	5

Input matrix (Shingles x Documents)

1	0	1	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0

Signature matrix *M*

2	1	2	1
2	1	4	1
1	2	1	2

Similarities:

	1-3	2-4	1-2	3-4
Col/Col	0.75	0.75	0	0
Sig/Sig	0.67	1.00	0	0

Min-Hash Signatures

- **Pick K=100 random permutations of the rows**
- Think of ***sig*(C)** as a column vector
- ***sig*(C)[i] =** according to the *i*-th permutation, the index of the first row that has a 1 in column *C*

  $$\boldsymbol{sig}(C)[i] = \min(\pi_i(C))$$

- **Note:** The sketch (signature) of document *C* is small **$\sim$100 bytes!**

- **We achieved our goal! We "compressed" long bit vectors into short signatures**

Costruzione di Firme Min-Hash

La slide descrive la costruzione di **firme Min-Hash**, una tecnica per comprimere rappresentazioni di dati (come documenti o colonne) mantenendo la capacità di stimarne la similarità.

Generazione delle Firme

Si scelgono $K = 100$ permutazioni casuali delle righe di una matrice. Per ogni permutazione π_i, la firma $\text{sig}(C)$ di una colonna C è un vettore colonna.

Calcolo degli Elementi della Firma

Per la i-esima posizione $\text{sig}(C)[i]$, si determina l'indice della prima riga che ha un 1 in C secondo la permutazione π_i. Formalmente:

$$\text{sig}(C)[i] = \min(\pi_i(C)),$$

dove min indica il valore minimo di indice dopo la permutazione.

Compattezza

La firma risultante (*sketch*) di un documento C è molto piccola, circa 100 byte, grazie all'uso di un numero limitato di permutazioni.

Obiettivo Raggiunto

Questo approccio consente di "comprimere" lunghi vettori di bit (rappresentazioni originarie dei dati) in firme brevi, mantenendo informazioni utili per confronti di similarità.

In Sintesi

Il **Min-Hash** crea firme compatte (100 permutazioni) calcolando gli indici minimi per ogni permutazione, raggiungendo l'obiettivo di ridurre la dimensionalità dei dati senza perdere la capacità di stimarne la similarità.

Implementation Trick

- **Permuting rows even once is prohibitive**
- **Row hashing!**
  - Pick **K = 100** hash functions k_i
  - Ordering under k_i gives a random row permutation!
- **One-pass implementation**
  - For each column C and hash-func. k_i keep a "slot" for the min-hash value
  - Initialize all $sig(C)[i] = \infty$
  - **Scan rows looking for 1s**
    - Suppose row j has 1 in column C
    - Then for each k_i :
      - If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

How to pick a random hash function h(x)?
Universal hashing:
$h_{a,b}(x)=((a \cdot x+b) \bmod p) \bmod N$
where:
a,b … random integers
p … prime number (p > N)

Trucco di Implementazione per Min-Hash

La slide presenta un trucco di implementazione per rendere efficiente il calcolo delle firme **Min-Hash**, evitando permutazioni costose e utilizzando un approccio basato su hashing.

Problema delle Permutazioni

Permutare le righe, anche una sola volta, è computazionalmente proibitivo a causa del loro alto numero.

Soluzione con Row Hashing

Si scelgono $K = 100$ funzioni hash k_j. Ogni funzione k_j genera una permutazione casuale delle righe, evitando la necessità di permutazioni fisiche.

Implementazione in un Passaggio (One-Pass)

Per ogni colonna C e funzione hash k_j:

- Si mantiene uno "slot" per il valore Min-Hash, inizializzato a ∞.
- Si scansionano le righe cercando 1:
  - Se la riga j ha un 1 in C, per ogni k_j, se:

    $$k_j(j) < \text{sig}(C)[j],$$

    si aggiorna:

    $$\text{sig}(C)[j] \leftarrow k_j(j).$$

Scelta delle Funzioni Hash

Si usa **universal hashing** per generare funzioni casuali. La formula è:

$$h_{a,b}(x) = ((a \cdot x + b) \mod p) \mod N,$$

dove:

- a, b: interi casuali,
- p: numero primo ($p > N$),
- N: dimensione dello spazio.

In Sintesi

Il trucco sostituisce le permutazioni fisiche con K funzioni hash casuali, implementando un algoritmo in un solo passaggio che aggiorna i valori Min-Hash scansionando le righe. L'**universal hashing** viene usato per generare permutazioni efficienti.

Row	C_1	C_2	C_3	C_4	x+1 mod 5	3x+1 mod 5
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

- For each column ***C*** and hash-func. k_i keep a "slot" for the min-hash value
- Initialize all ***sig(C)[i]*** = ∞
- **Scan rows looking for 1s**
  - Suppose row ***j*** has 1 in column ***C***
  - Then for each k_i:
    - If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

	C_1	C_2	C_3	C_4
h_1	1	3	0	1
H_2	0	2	0	0

Esempio Pratico di Calcolo delle Firme Min-Hash

La slide illustra un esempio pratico di calcolo delle firme **Min-Hash** per una matrice binaria, utilizzando due funzioni hash per stimare la similarità tra colonne.

Matrice Iniziale

La matrice di partenza è la seguente:

Row	C_1	C_2	C_3	C_4	$x+1 \mod 5$	$3x+1 \mod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

La tabella ha sei colonne e cinque righe di dati, più una riga di intestazione. Le intestazioni delle colonne sono: "Row", C_1, C_2, C_3, C_4, $x+1 \mod 5$, e $3x+1 \mod 5$. La prima colonna contiene i numeri da 0 a 4. Le colonne da C_1 a C_4 contengono per lo più 0 e 1. Le ultime due colonne contengono i valori precalcolati delle funzioni hash $h_1(x) = (x+1) \mod 5$ e $h_2(x) = (3x+1) \mod 5$.

Funzioni Hash

Si usano due funzioni hash per simulare permutazioni:

$$h_1(x) = (x+1) \mod 5,$$

$$h_2(x) = (3x+1) \mod 5.$$

I valori delle funzioni hash sono precalcolati per ogni riga (0-4) e mostrati nelle ultime due colonne della matrice.

Calcolo delle Firme

Per ogni colonna (C) e funzione hash k_j:

- Si inizializza $\text{sig}(C)[j] = \infty$.
- Si scansionano le righe cercando 1:
  - Se la riga j ha 1 in C, si calcola $k_j(j)$.
  - Se $k_j(j) < \text{sig}(C)[j]$, si aggiorna:

$$\text{sig}(C)[j] \leftarrow k_j(j).$$

Esempio per C_1

- h_1: Righe con 1 sono 0 e 3.

$$h_1(0) = 1, \quad h_1(3) = 4. \quad \text{Minimo: } 1.$$

- h_2:

$$h_2(0) = 1, \quad h_2(3) = 0. \quad \text{Minimo: } 0.$$

- Firma di C_1: $([1, 0])$.

Risultato

Le firme per le colonne sono rappresentate nella seguente matrice:

	C_1	C_2	C_3	C_4
h_1	1	3	0	1
h_2	0	2	0	0

Ovvero:

- C_1: $([1, 0])$,
- C_2: $([3, 2])$,
- C_3: $([0, 0])$,
- C_4: $([1, 0])$.

In Sintesi

La slide mostra come calcolare le firme Min-Hash con due funzioni hash, scansionando le righe per trovare i minimi valori hash delle righe con 1, ottenendo firme compatte per ogni colonna.