EditDistance / NGram

Pandolfini Luca

Settembre 2020

1 Introduzione

In questa relazione andrò a studiare il funzionamento dell'algoritmo di Edit-Distance, e come si possa utilizzare per trovare parole vicine ad una query Q in un lessico L, eseguendo un insieme di test che ci permettano di comprendere vantaggi e svantaggi dell'utilizzo di indici di n-gram per eseguire le query

2 Problema

Edit-distance è un algoritmo che quantifica la differenza tra due stringhe, contando il numero minimo di operazioni richieste per trasformare una stringa in un'altra, il costo di questa operazione è:

$$\Theta(m * n) \tag{1}$$

Dove m e n sono le lunghezze delle due stringhe, dato in Input Q sarà necessario eseguire l'algoritmo di edit-distance ogni stringa di L

3 Prestazioni Attese, Struttura

In questa relazione andrò ad utilizzare un lessico di parole italiane composto da 60000 stringhe.
Tenendo conto che la lunghezza media di una parola italiana è di 6 caratteri mi aspetto 60000 esecuzioni di edit-distance su stringhe, in media, di 6 caratteri, mentre per l'utilizzo di n-gram ho creato più dizionari che hanno come indici sillabe (2-gram) o triplette (3-gram), e come valori, liste di parole contenenti gli indici stessi, in questo caso invece, mi aspetto tempi di esecuzione in runtime ridotti, dipendenti dalla lunghezza delle liste su cui viene eseguita la edit-distance (le liste scelte saranno quelle con gli indici in comune con la query)
es. dizionario:

- "**aba**": ["sill**aba**", "dat**aba**se"];
- "**sto**": ["abbru**sto**lisco", "ago**sto**", "angu**sto**", "antipa**sto**", "apo**sto**li"]
- "**cer**":["**cer**care", "**cer**co", "ac**cer**tare"];

4 Test

I test che andrò ad eseguire non saranno altro che un confronto tra i due dizionari, 2-gram e 3-gram in relazione al tempo per generarli e alla memoria occupata, sono presenti due grafici per comprendere la distribuzione della loro tabella hash, indicata in blue la percentuale di indici contenenti liste con un numero pari o inferiore alle 1000 stringhe; per ultimo una tabella che mostra i tempi di ricerca di stringhe in L simili a Q.

	MB	sec	index < 1000
2-gram	5,14	13	58,4%
3-gram	4,54	300	97,7%

in blue percentuale chiavi con liste fino a 1000 elementi

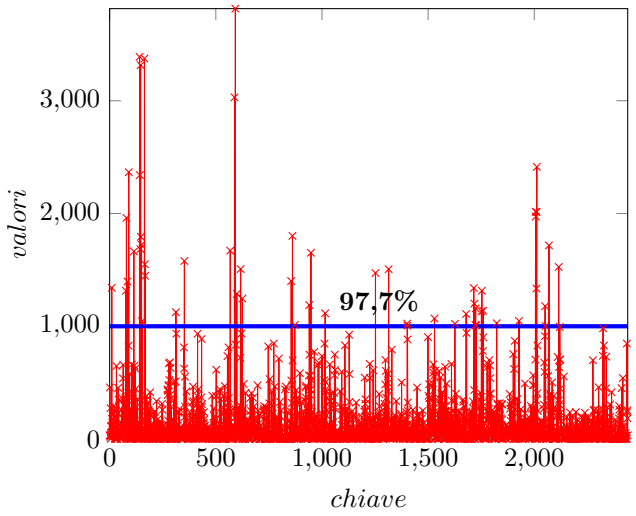

Figure 1: Distribuzione Dizionario 3-Gram

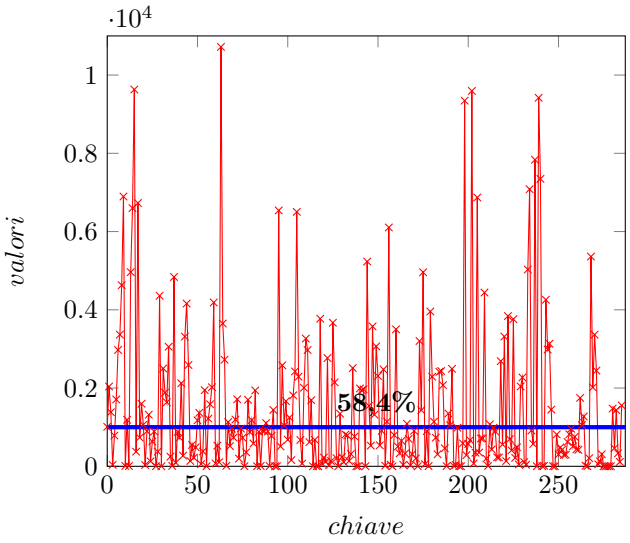

Figure 2: Distribuzione Dizionario 2-Gram

'parola'	2-gram	3-gram
cosa	0,28	0,020
anno	0,40	0,047
uomo	0,20	0,020
giorno	0,55	0,047
volta	0,46	0,039
casa	0,38	0,021
parte	0,60	0,045
vita	0,39	0,045
tempo	0,48	0,021
donna	0,42	0,020
mano	0,44	0,095
occhio	0,34	0,091
ora	0,31	0,017
signore	1.1	0,050
paese	0,47	0,010
elettrodomestico	4.4	0,67
bere	0,9	0,18
cezcaze	0,5	FAILED

5 Conclusioni

Come si può notare dai test eseguiti, la distrubuzione delle parole nel dizionario è migliore in quello da 3-gram, genera meno collisioni ed essendoci solo il 2,3%

di indici con collisioni sopra le 1000 stringhe, esegue in media edit-distance su molte meno stringhe rispetto al dizionario 2-gram.

Non sono presenti i test effettuati sulla edit-distance su tutto il dizionario in quanto i tempi di esecuzione non dipendono dalla parola inserita ma soltanto dal numero di caratteri di Q, come indicato dal suo costo: eq (1)

Gli unici due svantaggi di utlizzare il dizionario 3-gram sono che, il tempo per essere generato è 30 volte superiore a quello di 2-gram, e che è "meno" preciso del 2-gram ad esempio se effettuo un errore in ogni tripletta non troverà mai la parola che in realtà volevo intendere, a differenza del 2-gram: es. la parola 'cezcaze' intendendo cercare, la 3-gram non la trova la 2-gram si.

Degna di nota anche la parola 'bere' che nonostante la sua lunghezza i suoi tempi sono più alti perchè contenente la tripletta 'ere' che è l'indice con più collisioni (3817, in 3-gram)

Una ultima accortezza prima di concludere: bisogna tener conto del fatto che la responsabilità del numero elevato delle collisioni non sia dato dalle radici delle parole ma dalle coniugazioni della lingua italiana (es. are, ere, ire, ato, ito), con qualche modifica del sistema di ricerca, ad esempio effettuato a priori senza coniugazione ed in un secondo tempo riproposto con le varie declinazioni, creerebbe hash più uniformi, e quindi ricerche più veloci.