Algoritmi di calcolo di Edit-distance e di indici di n-gram per confronti tra parole

Andrea Spitaleri

July 3, 2018

1 Introduzione

Gli algoritmi che vediamo oggi hanno molteplici impieghi in tutti quei software che hanno in un modo o nell'altro a che fare con parole. Sono parte integrante degli algoritmi di correzione di errori di ortografia di ogni suite di software da ufficio, o di ogni motore di ricerca online. In sintesi tutto quello di cui si occupano è creare, a partire da una data parola e da un dizionario, una lista di parole grammaticalmente simili a quella data. Ciò è particolarmente utile poiché ad esempio, se un utente inserisce, in un documento di testo, una parola con un errore ortografico, il sistema può venirgli incontro fornendo delle interpretazioni adeguate della parola erroneamente digitata. Vediamo dunque come funzionano questi due algoritmi, e in cosa si differenziano.

2 Edit-distance

Questo algoritmo incarna il concetto di programmazione dinamica: un tipo di programmazione in cui un problema viene ridotto in più parti atomiche, sottoproblemi le cui soluzioni vengono annotate in una tabella al fine di ottimizzare ed evitare ridondanze nel calcolo dei problemi via via più grandi, fino a trovare la soluzione del problema principale. Edit-distance, infatti, date due parole, genera una tabella nella quale sono memorizzati i costi per trasformare una parola X in un'altra parola Y. Le operazioni concesse sono elementari: inserimento, cancellazione, sostituzione di una lettera, o scambio di posizione di due lettere consecutive. Da questa tabella possiamo ricavare il costo per trasformare X in Y ("distanza" tra X e Y). Ad esempio la distanza tra Cavallo e Cammello è 3 (due sostituzioni e un inserimento).
Le tabelle generate da questo algoritmo sono del tipo:

		b	e	l	l	o
	0	1	2	3	4	5
b	1	0	1	2	3	4
r	2	1	1	2	3	4
i	3	2	2	2	3	4
l	4	3	3	2	2	3
l	5	4	4	3	2	3
o	6	5	5	4	3	2

La cifra in posizione i,j indica quanto costa (in termini delle sopraelencate operazioni elementari) trasformare il prefisso, fino alla j-esima lettera, della parola X, nel prefisso, fino alla i-esima lettera, dell parola Y. Questi valori sono generati secondo un' intelligente dinamica:
Poniamo di voler calcolare il costo in pos. [3][3] (corrisponde al costo per trasformare "be" in "br"). L'algoritmo calcola le seguenti possibilità, e sceglie la meno costosa:

- Se la lettera alla quale si riferisce la pos. [3][3] è la medesima su X e su Y, allora ho come costo il medesimo di quello in [2][2] (infatti se non devo fare cambiamenti, non devo modificare nulla e quindi non compio operazioni). Non è questo il nostro caso, infatti abbiamo "e" ed "r".

- Se invece la lettera in pos. [3][3] non è la medesima su X e su Y (è questo il caso) allora al costo di [2][2] dobbiamo aggiungere 1 (ossia il costo per effettuare una sostituzione). In questo caso questo costo sarebbe 1, perché è dato dal costo per trasformare "b" in "b", ovvero nulla, più il costo per la sostituzione da "e" a "r".
- Consideriamo il costo per una eventuale cancellazione, dato dal costo in [2][3] +1. In questo caso sarebbe 2.
- Il costo per un inserimento. In questo caso sarebbe 2.
- Infine (non è questo il caso) se la j-esima lettera di X è uguale alla (i-1)-esima di Y, e la (j-1)-esima di X è uguale alla i-esima di Y, allora possiamo scambiare le due lettere e costa soltanto 1. Quindi si dovrebbe considerare anche il costo in [i-2][j-2] +1.

Il minore tra questi costi è 1, ossia ciò che conviene fare è tenersi la "b" e cambiare la "e" in "r". L'ultimo valore della tabella indica il costo complessivo per passare da X a Y. In questo caso è 2, infatti ci bastano una sostituzione e un inserimento.
Viene spontaneo chiedersi quanto può costare, in termini computazionali, un algoritmo del genere, che cioè costruisce tante tabelle, secondo queste direttive, quante sono le parole di un intero dizionario. Se come dizionario prendiamo ad esempio quello della lingua italiana, già soltanto ad intuito si può essere portati a pensare che i tempi richiesti per fornire una lista di parole affini (poco distanti) a quella proposta siano improponibili. E l'intuito incontra la ragione, se valutiamo analiticamente tale costo: abbiamo infatti un costo di $\Theta(mn)$, con m lunghezza di X, e n lunghezza di Y, ma con una costante davvero molto alta, ossia compatibile col numero di parole contenute nel dizionario italiano. Quindi, peraltro, confrontare parole molto lunghe incide notevolmente sulle prestazioni.
Occorre perciò, se vogliamo confrontare con dizionari di queste dimensioni, adoperare un'altra soluzione. Con l'algoritmo che crea indici di n-gram adoperiamo un approccio diverso.

3 Calcolo di indici di n-gram

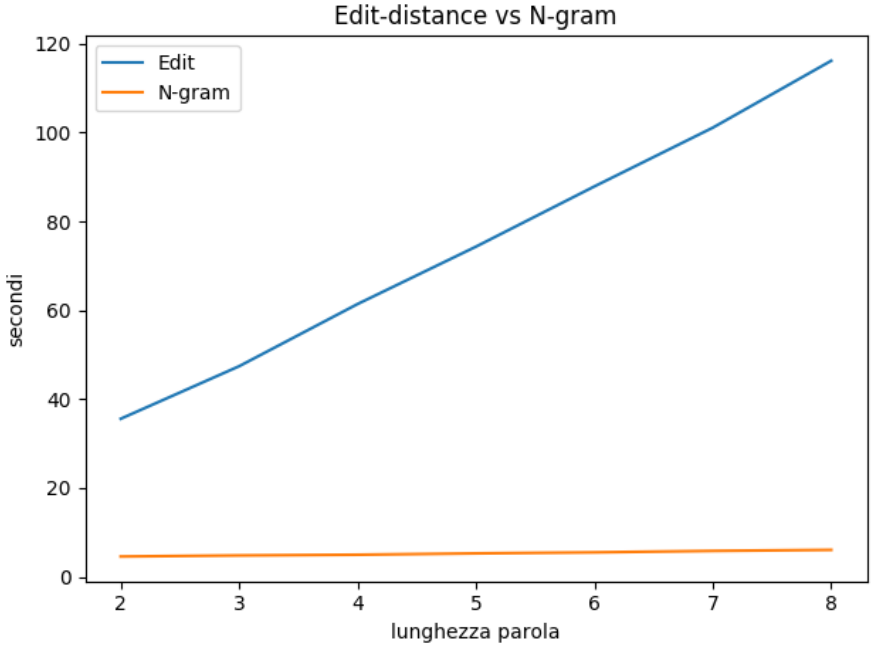
Questo approccio, invece di calcolare una effettiva e precisa distanza tra due parole, ne da una idealizzazione secondo un'altrettanto intelligente dinamica:
Questo algoritmo infatti, spezza le parole mettendole in un insieme contenente gruppi sovrapposti di n lettere (n-grammi. n consigliato tra 2 e 4). Ad esempio, la parola Isabella, con n = 3, diventerebbe un insieme contenente: isa, sab, abe, bel, ell, lla. Con n = 4, diventerebbe: isab, sabe, abel, bell, ella.
Ciò viene fatto con la parola data, e, di nuovo, con tutte le parole in un dizionario. Definisco il coefficiente di Jackard: questo valore è quello che da una stima di quanto due parole siano affini. Dati gli insiemi di n-grammi appartenenti a due parole, esso è dato dal rapporto tra la cardinalità dell'intersezione dei due insiemi fratto la cardinalità dell'unione dei due insiemi. In soldoni, è il rapporto tra il numero di n-grammi in comune, fratto il numero totale di n-grammi senza ripetizioni o doppioni. Tale coefficiente, per come definito, vale 1 per due parole identiche, e 0 per parole totalmente differenti. Si inizia a considerare due parole come affini, se il loro coefficiente di Jackard supera lo 0.7 - 0.8 .
Con una scelta di n = 2 avremo più risultati, più precisi e attinenti, al costo di un tempo di esecuzione lievemente (ma davvero di un niente) maggiore. Con una scelta di n = 4 di controcanto, avremo meno risultati, ma saranno erogati in tempi lievemente minori (davvero di un niente).

4 Analisi prestazionali

Risulta logico pensare che la suddivisione di una parola e il calcolo di un coefficiente siano computazionalmente più digeribili, piuttosto che la creazione di una intera tabella. Infatti il costo per il calcolo degli indici di n-gram è molto minore del costo di Edit-distance. Probabilmente, computazionalmente parlando, incide maggiormante il calcolo del coefficiente di Jackard che la suddivisione vera e propria delle parole, tant'è vero che il calcolo di tale coefficiente richede il confronto tra elementi di due insiemi. Ciò è comunque estremamente meno impegnativo sulla CPU che il calcolo di intere tabelle.

5 Sperimentazione

Si rende utile impostare come oggetto della sperimentazione la differenza tra i tempi di esecuzione di Edit-distance e N-gram al variare della lunghezza della parola data. La lunghezza della parola data è un cruciale discriminante al fine di evidenziare la differenza tra questi due algoritmi.
Mi sono servito di un dizionario che offre un' ampia scelta di parole italiane, realizzato dall'utente Github Francesco Napoletano, il quale offre liberamente diverse versioni del dizionario, a seconda di svariate esigenze. Il dizionario che ho utilizzato per questo test in particolare, comprende un totale di circa 660000 parole italiane. Ho condotto una decina di test, di cui riporto le medie dei tempi di esecuzione di entrambi gli algoritmi. Il test esegue Edit-distance ed N-gram su parole scelte a caso, ma di lunghezza via via crecente (da parole di 2 lettere, a parole di 8 lettere). In particolare ho scelto come parametri: per Edit-distance ho impostato una distanza di edit di 1: l'algoritmo di volta in volta cerca parole distanti al massimo 1 dalla parola data; per N-gram ho impostato n = 2, e coefficiente di Jackard = 0.65: l'algoritmo genera bi-grammi, e ritiene che due parole siano simili se il loro coefficiente di Jackard è superiore allo 0.65 (che è tutto sommato piuttosto concessivo).
I risultati osservati sono i seguenti:



	2	3	4	5	6	7	8
Edit-distance	35.58	47.41	61.41	74.33	87.89	101.12	116.19
N-gram	4.56	4.81	4.97	5.27	5.50	5.83	6.06

Si rende oltremodo evidente anche in questo esercizio, la differenza abissale tra i tempi medi di esecuzione, e quanto questa differenza cresca all'aumentare della lunghezza della parola data. Ciò è in totale accordo con quanto anticipato prima. La lunghezza della parola data è il fattore che influenza maggiormente i tempi di esecuzione di entrambi gli algoritmi, come possiamo notare dalla tabella, ma incide in maniera più importante sull'algoritmo Edit-distance, piuttosto che su N-gram, nel quale i tempi di esecuzione rimangono decisamente contenuti.

6 Conclusioni

Edit-distance è un algoritmo di tutto rispetto se utilizzato con dizionari di "pochi" elementi, perché è più preciso nel valutare quali parole siano effettivamente più simili a quella data, o almeno questo è quello che si rende evidente nei test effettuati, condotti a partire dal file Python in allegato. Il calcolo degli indici di n-gram invece è a dir poco fulmineo, e trova parole piuttosto simili a quella

data, ma i risultati possono talvolta non includere, o essere lievemente diversi da quelli di Edit-distance. I suoi tempi di esecuzione, però, a mio modesto parere, fanno pendere l'ago della bilancia a favore di N-gram, perché tra i due è l'unico che può fornire risultati utili in tempi umanamente tollerabili.
Andrea Spitaleri