Edit Distance

di Pagliocca Marco

1 Introduzione

Con "distanza" tra due stringhe si intende il minor numero di operazioni strettamente necessarie a trasformare la stringa X nella stringa Y. Le operazioni a cui si fa riferimento possono essere di vario genere: inserimento, cancellazione, copia, scambio tra due lettere etc...
La soluzione dell'Edit Distance permette di affrontare problemi ben piú interessanti, quali la correzione ortografica o di documenti. Di fatto sará tentata la realizzazione di un algoritmo in grado di trovare i nomi propri piú 'vicini' ad una data query Q.

1.1 Edit Distance

La soluzione per l'Edit Distance puó essere formulata in modo ricorsivo mediante i sotto-problemi:

"Trovare la sequenza minima di operazioni necessarie per convertire X_i in Y_j, supponendo di conoscere l'ultima operazione utilizzata."

Dopo aver definito sia il problema che la distanza, resta da delineare quali operazioni utilizzare e quale sia il loro costo. Le scelte implementabili possono essere variegate, tuttavia le principali, da prendere in considerazione, sono:

Copia delle ultime lettere di ciascuna stringa, nel caso siano uguali. Viene implementata a costo nullo;

Sostituzione dell'ultima lettera di X con l'ultima lettera di Y. Quando le due lettere sono diverse viene implementata a costo unitario;

Scambio di due coppie di lettere uguali ma in posizioni alternate. Servendosi di una variabile di appoggio é implementabile con 3 operazioni elementari;

Inserimento di una nuova lettera nella stringa X. É un'operazione unitaria;

Cancellazione di una lettera dalla stringa X. Anche questa a costo unitario.

Utilizzando una matrice c di dimensione (N+1)x(M+1) (N=lunghezza di X, M=lunghezza di Y), l'elemento c[i, j] rappresenta il costo per trasformare X_i in Y_j. Si ha che:

$$c[i,j] = \begin{cases} c[i-1,j-1] + costo(copia) & \text{se } X_i = Y_j \\ c[i-1,j-1] + costo(sostituzione) & \text{se } X_i \neq Y_j \\ c[i-2,j-2] + costo(scambio) & \text{se i,j} \geq 2 \text{ e } X_i = Y_{j-1} \text{ e } X_{i-1} = Y_j \\ c[i-1,j] + costo(cancellazione) & \text{sempre} \\ c[i,j-1] + costo(inserimento) & \text{sempre} \end{cases}$$

Il costo minimo, in termini di operazioni, necessario per trasformare X in Y é l'ultimo elemento della matrice, ovvero c[N, M]. Per raggiunge il valore ottimo é necessario prima risolvere tutti i sotto-problemi che lo precedono. Ogni sotto-stringa di X viene confrontata con ogni sotto-stringa di Y, dunque il tempo di esecuzione asintotico é $\theta(N * M)$.
Le risorse impiegate non si limitano al tempo, ma anche allo spazio di memorizzazione. Anche in questo caso é $\theta(N * M)$.

1.2 Indici di n-gram

Il costo di Edit Distance viene utilizzato per sapere se una parola é piú vicina ad una query Q rispetto ad altre parole di un lessico L. Tuttavia, a causa del tempo di esecuzione elevato, calcolare la distanza dalla query per ogni parola del lessico é un'operazione onerosa. Per ridurre il numero di parole da confrontare si puó utilizzare gli **indici di n-gram**.
Si procede in due modi:

- trovo tutti i termini in L che contengono qualunque n-gram di Q;
- cerco tutti i termini in L che contengono abbastanza n-gram di Q.

In entrambi i casi ottengo un sotto-insieme di L e riduco notevolmente i tempi di esecuzione.
Nel secondo caso devo considerare il **coefficiente di Jaccard**, ovvero un modo per determinare quanto due insiemi contengano elementi uguali indipendentemente dalla loro dimensione:

$$JC = \frac{|J \cap C|}{|J \cup C|}$$

dove J e C rappresentano gli insiemi di n-gram della stringa X e quella Y.
Poi si considera una soglia minima e arbitraria, sotto la quale la parola non viene presa in considerazione: deve essere un valore non negativo e minore di 1; una buona scelta é tra 0.6 e 0.8, che corrisponde ad avere parole con 1/2 n-gram di differenza.
In questo contesto é importante affermare che la dimensione n degli n-gram é importante ai fini della valutazione, poiché valori troppo piccoli (al piú 1) possono generare insiemi molto grandi, mentre valori troppo grandi non permettono di ottenere una selezione adeguata. Una scelta giusta puó essere in funzione della dimensione delle stringhe.

2 Implementazione su Python

Edit Distance é un algoritmo piuttosto lineare per cui non servono grossi accorgimenti nella sua implementazione. Richiede come input due stringhe X e Y di dimensione N e M, dalle quali viene creata una matrice c di dimensione (N+1)x(M+1) utile a contenere le soluzione ottime di tutti i sotto-problemi. L'inizializzazione definisce le operazioni dei casi base:

- c[0,j] = j*costo(inserimento)
- c[i,0] = i*costo(cancellazione)
- c[0,0] = 0

mentre il corpo centrale calcola iterativamente ogni altra soluzione nelle modalitá precedentemente esposte. Anche le operazioni eseguite e i costi sono i medesimi.
Un particolare del problema riguarda l'utilizzo di una seconda matrice, utile a contenere le operazioni eseguite per la trasformazione. Poiché ci interessa

unicamente la distanza tra le stringhe, cioé il valore ritornato, e non le operazioni per ottenerla, la matrice *op* verrá inserita per completezza, ma sotto forma di commento.

Indici di n-gram sono il nucleo del problema e richiedono un'implementazione un pó piú articolata.
La prima scelta riguarda la dimensione degli n-gram. Sulla base della precedente analisi, una funzione ausiliaria *chooseGram* riceve in ingresso una stringa, ne calcola la lunghezza e vi definisce il valore di *numGram* come la parte intera inferiore del logaritmo in base 2. Va da sé che per parole corte si otterranno n-gram di piccole dimensioni (ma almeno di 2), mentre all'aumentare della lunghezza, anche gli n-gram saranno sempre piú grandi, ma seguendo un andamento logaritmico.
ChooseGram chiama dunque la funzione *nGram* che restituisce la lista degli n-gram della stringa X. Nel caso in cui *numGram* sia maggiore o uguale della lunghezza, allora viene restituita la stringa intera. Nel caso generico, invece, un ciclo scorre nella stringa copiando gli n-gram di dimensione *numGram* fino ad arrivare in fondo.

Calcolatore dei nomi (*nameCalculator*) viene utilizzato per ricavare gli *numWords* nomi propri, prelevati dalla lista *nearestNames*, piú prossimi ad una query. A tal proposito sfrutta:

- una lista (*probableNames*) contenente le parole e il cui ordine indica l'importanza;
- un dizionario (*ordWords*) che indica la classifica delle parole trovate rispetto alla distanza di editing dalla query.

Per ogni parola in *nearestName* calcola l'edit distance con la query e ne confronta il valore con quelli precedentemente calcolati. Fin quando il numero di parole inserite rimane inferiore a quelle cercate é sicuro che la nuova parola venga aggiunta alla lista; dopodiché l'inserimento avviene se e solo se la nuova parola é piú vicina alla query rispetto (almeno) all'ultima parola in *probableName*.

3 Gli esperimenti

Gli esperimenti devono permettere di comprendere vantaggi e svantaggi dell'utilizzo di indici di n-gram per eseguire le query.
Alla luce della teoria, sembrerebbero portare soli vantaggi: limitano il calcolo dell'edit distance alle sole parole 'vicine' alla query. In realtá gli esperimenti mostrano l'altra faccia degli n-gram: la "vicinanza" intesa dagli n-gram é differente da quella dell'edit distance, ovvero le parole filtrate non é detto che siano le meno distanti, dove con 'distanza' si rimanda alla definizione data nel capitolo 1.
Detto questo gli esperimenti verranno eseguiti su dati uguali, ma prima senza e poi con l'ausilio degli n-gram, e si articoleranno come segue:

1. Viene fornita una lista di 4 nomi raccolti effettuando varie prove che permettono di focalizzare l'attenzione su vari aspetti degli algoritmi testati:

   Abbondanzio mostra come il tempo di esecuzione aumenta in modo vertiginoso all'aumentare della dimensione della stringa;

   Marca mette in luce gli svantaggi nell'uso degli n-gram;

   Luha, distorsione di Luca, permette di testare la funzionalitá dell'edit distance;

   Anna mostra molte varianti del nome rispetto alle due modalitá implementate.

2. Viene aperto e convertito in lista un file di testo (*9000nomipropri.txt*) contenente circa 9000 nomi propri: sará il lessico su cui implementare gli algoritmi. É stato scaricato dal sito web *https://github.com/napolux/paroleitaliane*;

3. Per ogni query, viene chiamato 5 volte *nameCalculator* fornendo come lessico il vocabolario intero. Ad ogni iterazione si misura il tempo necessario all'esecuzione come differenza del tempo registrato prima e dopo la chiamata a funzione, e si sommano i tempi in un dizionario relativo alla query in uso. Al termine si divide il valore per 5 ottenendone la media;

4. Per ogni query, viene chiamato 5 volte una funzione filtrante *jaccardGram* che calcola le parole che contengono abbastanza n-gram della

query, seguito dalla chiamata a *nameCalculator* col lessico filtrato. Le misurazioni temporali iniziano prima della chiamata a *jaccardGram* e terminano dopo la chiamata a *nameCalculator*. Viene usato un altro dizionario per tenere in memoria i tempi medi.

Un ultimo dettaglio importante: in *jaccardGram* si decide se inserire o meno la parola nella lista sulla base del coefficiente di Jaccard. Ebbene la dimensione dell'intersezione degli insiemi di n-gram delle stringhe confrontate equivale alla loro **sotto-sequenza piú lunga**; mentre l'unione viene implementata mediante la funzione nativa di Python *set().union()*. Poiché l'unione non contiene eventuali gram ripetuti, il coefficiente di Jaccard potrebbe assumere un valore maggiore di 1.

Specifiche del calcolatore: gli esperimenti vengono eseguiti su un *ASUS Laptop X541UV* a 8GB di memoria RAM e con processore Intel Core i7-6500U a 2.50GHz di frequenza. L'architettura a 64-bit ospita come Sistema Operativo Windows 10 Home.

Dettagli implementativi aggiuntivi: per ottenere risultati adeguati é sufficiente utilizzare un'approssimazione alla quarta cifra non nulla dei tempi misurati.
Inoltre gli esperimenti si basano sul numero di parole da ricercare: imponendo a 7 tale valore si possono mettere in luce differenze relative alle due modalitá eseguite.
Infine é stato scelto come soglia per i coefficienti di Jaccard un valore di 0.6, inerente alla spiegazione data nel capitolo 1.

4 Risultati dei Test

Per una completa visione dei risultati si rimanda al file *Output.txt* allegato. Verranno citati i risultati di maggior interesse.

Marca I risultati ottenuti dall'applicazione diretta di edit distance mostrano come alcune delle parole piú vicine non si ottengono dall'utilizzo degli indici di n-gram. Si veda 'mara': i 2-gram in comune sono 'ma' e 'ar', cioé solo 2 rispetto ai 5 della loro unione, per cui si ottiene un JC = 0.4, inferiore alla soglia. A riprova della differenza concettuale di "distanza", la stringa

'mara' dista dalla query solamente 1 (cancellazione della lettera c), cosí come 'marco', peró quest'ultimo appare nella lista *Con n-gram*, mentre la prima non é presente. Eppure la lista avrebbe altri 5 posti liberi:

Senza n-gram	marca	mara	marco	marga	maria	marica	marna
Con n-gram	marca	marco	-	-	-	-	-

Anna Qualcosa di simile si ottiene pure con questa query:

Senza n-gram	anna	anca	anda	anno	anta	fanna	ianna
Con n-gram	anna	fanna	ianna	janna	manna	nanna	vanna

Tuttavia in questo caso ci sono sufficienti nomi vicini per entrambe le modalitá, per cui si vede che alcuni dei nomi presenti in *Senza n-gram*, quali 'anca', 'anda', 'anno' e 'anta', non ci sono nella seconda riga. Al loro posto vengono riportati altre parole vicine ad *anna*, ma piú distanti rispetto alla definizione del capitolo 1.
Pertanto lo svantaggio principale nell'uso degli n-gram consiste nell'eliminazione di alcune parole dal lessico che sarebbero delle buone alternative alla query.

Luha L'idea che sta dietro alla scelta di questa query é che un programma implementante l'edit distance dovrebbe riuscire a fornire le parole scritte correttamente. In effetti l'algoritmo di per sé permette di ottenere la sostituzione piú immediata, cioé Luca, come parola piú vicina.
Tuttavia l'edit distance con n-gram fornisce come risultato un insieme vuoto, il che significa che non ci sono parole fornite da *nearestNames*. Invece di dare alternative meno giuste, in questo caso l'uso degli indici di n-gram ha determinato un errore piuttosto grave, non riuscendo a trovare nemmeno una sostituzione valida. Si veda, invece, nella Tabella 4 quante parole sarebbe possibile mostrare.

Senza n-gram	luca	luna	aura	eura	lapa	lara	laura
Con n-gram	-	-	-	-	-	-	-

Tabella 4: Risultati dell'edit distance della query Luha

Abbondanzio Come giá menzionato nel capitolo 3, l'uso di una query molto piú lunga aumenta vertiginosamente il tempo necessario per calcolare la distanza con le parole. Infatti dipendendo dalla lunghezza delle stringhe, a paritá di lessico, una query piú lunga necessita di tempi piú lunghi. L'analisi é confermata dai risultati riportati nel grafico di Figura 1.

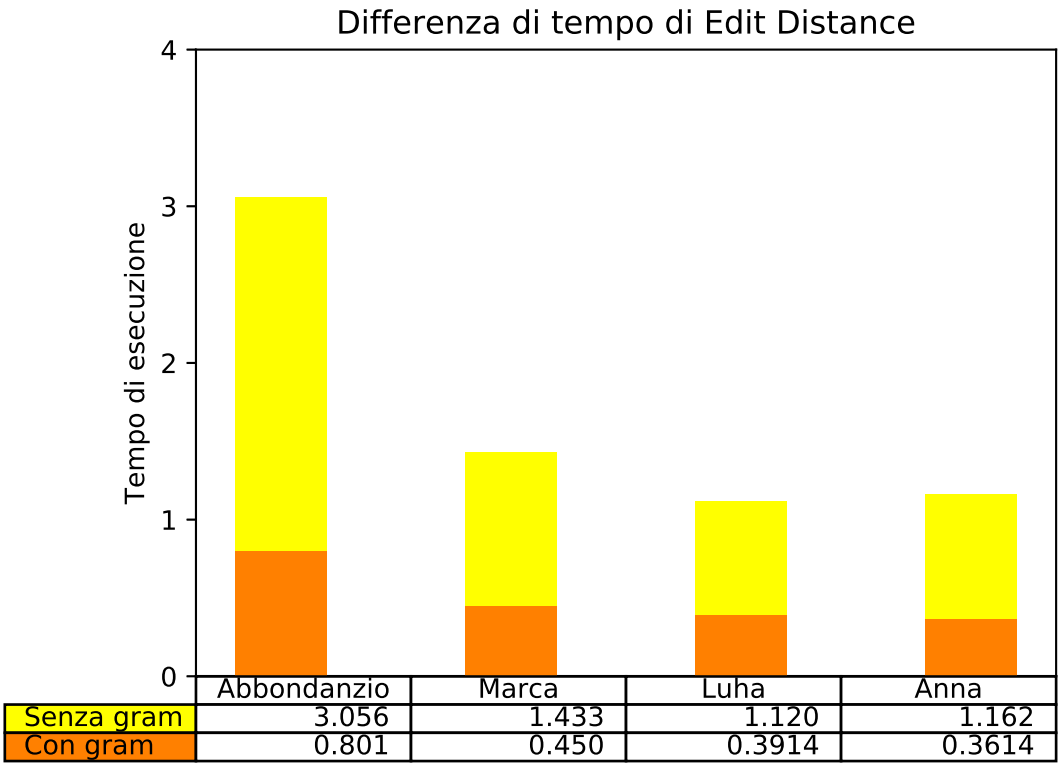

Figure 1: Differenze temporali ottenute dall'esecuzione dell'Edit Distance su query differenti, al variare dell'uso degli n-gram

La consistente differenza temporale viene messa in risalto dal grafico e colorata in giallo. Ad una prima approssimazione il costo per l'uso dell'Edit Distance senza indici di n-gram é il triplo rispetto al loro uso. Inoltre, il

tempo aumenta in modo considerevole all'aumentare della lunghezza della query: 'Luha' e 'Anna', che hanno entrambi lunghezza 4, presentano tempi simili; una sola lettera in piú, come in 'Marca' genera una crescita temporale per entrambe le modalitá. Se quindi si considera 'Abbondanzio' l'aumento é ben evidente.
I tempi registrati sotto al grafico corrispondo ai valori medi, mentre quelli parziali sono conservati in *Output.txt*.

5 Conclusioni

Edit Distance é l'algoritmo in grado di determinare la distanza tra due stringhe, ma é molto lento e necessita di euristiche interne per ridurre la dimensione del lessico. A tal proposito gli indici di n-gram permettono di filtrare le sole parole che hanno un certo numero di n-gram uguali alla query. Tuttavia non é un metodo del tutto affidabile poiché potrebbe eliminare dal confronto anche stringhe ad essa vicine, in quanto utilizza una definizione di "distanza" differente. D'altronde senza nessun accorgimento i tempi di esecuzione di Edit Distance lieviterebbero e l'algoritmo non potrebbe essere utilizzato in programmi che richiedono determinati tempi di risposta, come quelli di correzione ortografica (si pensi alla scrittura di un messaggio).
Per concludere, si potrebbe utilizzare euristiche interne di vario genere e in un determinato ordine, in modo tale da trovare la parola piú vicina alla query in periodi accettabili.