Edit distance e ricerca della parola più vicina

Diego Biagini

1 Introduzione

Un'operazione che viene frequentemente eseguita in programmi di vario tipo è la seguente: data una parola sbagliata o incompleta, trovare la parola più vicina ad essa.
Esistono numerosi approcci per risolvere questo problema, tutti si basano sul confronto della parola voluta con un dizionario delle parole possibili.
Tra queste parole è possibile trovare quella più vicina attraverso la cosiddetta edit-distance, distanza di editing, più questa è piccola più le parole sono simili tra loro.
Eseguire questa operazione per ogni parola possibile non è molto raccomandabile data la mole di un dizionario.
Inoltre la ricerca della parola più vicina viene usata molto spesso in applicazioni real time, come suggerimenti di autocompletamento, è quindi necessario che abbia buoni tempi di esecuzione, anche minori di un secondo.
Per raggiungere questo obiettivo è necessario diminuire il numero di parole con cui eseguiremo edit-distance, garantendo però che la parola più vicina sia considerata tra questi confronti.

2 Cenni teorici

2.1 Edit distance

Date due sequenze di caratteri X e Y è possibile definire la distanza di edit tra esse come il minimo numero di operazioni elementari da eseguire su una delle due in modo che diventino uguali.
Le operazioni elementari sono le seguenti:

- Lasciare immutato un carattere
- Inserimento di un carattere
- Rimozione di un carattere
- Sostituzione di un carattere
- Scambio di due caratteri

Ognuna di esse ha un certo costo predefinito.
É possibile definire una sottostruttura ottima del problema.
Definiamo $X_i =< x_1, x_2, ..., x_i >$ e $Y_j =< y_1, y_2, ..., y_j >$ sottosequenze di X e Y.
Se $C_{i,j}$ è la soluzione ottima di $edit\text{-}distance(X_i, Y_j)$ essa contiene soluzioni ottime dei problemi precedenti.
Quindi possiamo definire C ricorsivamente in questo modo:

$$C_{i,j} = min \begin{cases} C_{i-1,j-1} + cost(copy) & \text{se} X[i] = Y[j] \\ C_{i-1,j-1} + cost(replace) & \text{se} X[i] \neq Y[j] \\ C_{i-2,j-2} + cost(twiddle) & \text{se } i,j \geq 2, X[i] = Y[j-1] \text{ e } X[i-1] = Y[j] \\ C_{i-1,j} + cost(delete) & \text{sempre} \\ C_{i,j-1} + cost(insert) & \text{sempre} \end{cases}$$

Usando questo approccio e salvando in una tabella i costi dei sottoproblemi, avremo che il costo di tempo e spazio per l'algoritmo è $\Theta(m \cdot n)$ dove m e n sono le lunghezze delle sequenze originali.

2.2 Intersezione di n-gram

Data una sequenza di caratteri e un valore n è possibile definire un n-gram della sequenza come una sua sottosequenza di n elementi.
Per esempio i 3-gram della parola "parola" sono: "par","aro","rol","ola".
Un utilizzo degli n-gram è quello di controllare se due parole sono relativamente vicine tra loro. Infatti siamo sicuri che se tra due parole non si hanno n-gram in comune allora difficilmente sono simili.

É possibile quantificare questa somiglianza attraverso il **coefficiente di Jaccard**.
Dati due insiemi X e Y il coefficiente di Jaccard fra i due è: $JC = \frac{|X \cap Y|}{|X \cup Y|}$
Se i due insiemi coincidono allora il coefficiente di Jaccard è pari ad 1.

3 Esperimenti svolti

Ciò che ci interessa verificare è quale modo di trovare la parola più vicina ad una query Q è migliore. In questo controllo sarà necessario tenere conto del tempo impiegato nella ricerca e se il risultato della ricerca è quello giusto.
I modi di trovare la parola più vicina presi in considerazione sono i seguenti:

- eseguire edit distance tra Q e tutte le parole nel dizionario, restituiamo quindi quella con distanza minore
- eseguire edit distance tra Q e le parole nel dizionario che hanno almeno un n-gram in comune con Q
- eseguire edit distance tra Q e le parole nel dizionario il cui coefficiente di Jaccard con Q è superiore a una certa soglia

Il dizionario usato contiene 95000 parole della lingua italiana compresi i nomi propri.
I test saranno eseguiti su un insieme di parole sbagliate scelte a caso, indicativamente 100.
Per ottenerlo saranno scelte un certo numero di parole dal dizionario, dopodichè modificheremo un numero casuale di caratteri in ognuna di esse.
La scelta di quanti caratteri modificare sarà fatta casualmente ma con un criterio:

- se la parola è lunga 3 o meno caratteri sarà modificato 1 carattere
- se la lunghezza della parola è compresa tra 4 e 5 caratteri ne saranno modificati 1 o 2
- se la lunghezza della parola è maggiore di 5 saranno modificati 1,2 o 3 caratteri.

Nel calcolare gli n-gram di una parola saranno aggiunti all'inizio e alla fine di essa degli spazi, in questo modo si può distinguere se due parole iniziano o finiscono nello stesso modo.
Eseguiremo poi i tre modi sopracitati sull'insieme delle parole sbagliate, verificando anche se la parola più vicina trovata corrisponde alla parola originale.
Per ogni metodo sarà registrato il tempo medio di esecuzione su tutte le parole e l'hit rate, ovvero la percentuale di parole indovinate.
Gli n-gram che saranno sperimentati saranno i 2-gram, 3-gram, 4-gram dato che valori superiori sono molto predisposti a errori.
Le soglie del coefficiente di Jaccard saranno invece 0.2,0.5 e 0.8.

Gli esperimenti saranno condotti su una macchina con sistema operativo Linux Ubuntu 18.04, 6 GB di memoria ram, una cpu Intel Core i5(2.50 GHz / 3.10 GHz, 2 core).

4 Documentazione del codice

Il progetto è diviso in 4 file:

- main.py contiene il programma di testing
- editdistance.py contiene le funzioni per calcolare l'edit distance
- ngram.py contiene le funzioni per ottenere ngram di una parola o di un dizionario e per trovare la parola più vicina usando metodi con ngram
- util.py contiene funzioni varie come il calcolo del coefficiente di Jaccard o di modifica di stringhe

`edit_distance_matrix(x, y)`

Prende in input due stringhe e esegue l'algoritmo per trovare edit distance tra le due, restituisce le matrici che sono risultato dell'algoritmo.

`edit_distance(x,y)`

Esegue edit distance tra x e y e restituisce solo la distanza tra le due.

`get_ngram(word, n)`

Prende come parametro una parola e n, restituisce tutti gli n-gram della parola in una lista.

`dictionary_ngram(dictionary, n)`

Prende come parametro una lista di parole e restituisce tutti gli n-gram di ogni parola nella lista originale.

`get_jaccard_value(set1, set2)`

Prende due insiemi(liste) e restituisce il coefficiente di Jaccard tra esse.

`closest_word(word, dictionary)`

Prende una parola e un dizionario. Restituisce la parola presente nel dizionario più vicina a quella passata controllando tutte le parole. Restituisce inoltre la distanza tra la parola trovata e quella passata.

`closest_word_ngram_ed_jacc(word,n,n_grammed_dictionary, jaccard)`

Prende una parola,il dizionario contenente gli n-gram di tutte le parole e una soglia per il coefficiente di Jaccard.
Calcola il coefficiente di Jaccard tra la parola e ogni parola nel dizionario, dopodichè esegue edit distance con le parole con coefficiente superiore alla soglia passata in ingresso e restituisce la parola più vicina tra queste.

`closest_word_ngram_ed_1gram(word,n, n_grammed_dictonary)`

Prende una parola e il dizionario contenente gli n-gram di tutte le parole.
Esegue edit distance con le parole che contengono almeno un n-gram in comune con la parola passata come parametro, restituisce poi quella più vicina.

`mistype_word(word, n)`

Prende come parametro una parola e un numero di caratteri. Restituisce la parola dopo aver sostituito n caratteri con altri scelti casualmente.

`mistype_list(words)`

Prende come parametro una lista di parole e cambia un numero casuale di lettere in ognuna di esse. Questo numero è scelto in base alla dimensione della parola: se è lunga meno di 3 caratteri sarà cambiato 1 carattere, se è lunga da 4 a 5 ne saranno cambiati 1 o 2, se è lunga più di 6 ne saranno cambiati 1,2 o 3.

5 Risultati sperimentali

Esecuzione della procedura di edit distance su tutto il dizionario:
Tempo medio di esecuzione: 43.66 s
Hit rate: 0.77

Table 1: Tempo medio di esecuzione(s)

	Almeno un n-gram	Coefficiente di Jaccard ≥ 0.2	Coefficiente di Jaccard ≥ 0.5	Coefficiente di Jaccard ≥ 0.8
2-gram	35.37	1.199	0.372	0.368
3-gram	5.65	0.411	0.324	0.322
4-gram	1.49	0.316	0.281	0.281

Table 2: Hit rate

	Almeno un n-gram	Coefficiente di Jaccard ≥ 0.2	Coefficiente di Jaccard ≥ 0.5	Coefficiente di Jaccard ≥ 0.8
2-gram	0.77	0.75	0.43	0.06
3-gram	0.77	0.63	0.31	0.04
4-gram	0.71	0.55	0.14	0.04

6 Analisi e conclusioni

La prima cosa che è possibile notare nell'analizzare la tabella 1 è la notevole velocità assunta da ricerche di parole più vicine attraverso una soglia del coefficiente di Jaccard.
Questa è guadagnata a scapito dell'accuratezza dell'algoritmo, infatti con una soglia superiore allo 0.8 è quasi impossibile trovare esattamente la parola che stavamo cercando(tabella 2) tra tutte le possibili, quindi è da escludere in un uso reale.

Altra cosa che è sicuramente da escludere è il semplice edit-distance con tutte le parole del dizionario, infatti con un tempo di esecuzione medio di 43 secondi è veramente difficile da usare, sia in applicazioni real time come suggerimenti di autocompletamento, sia in campi più lenienti come la correzione di documenti.
Esso è il caso ottimo in fatto di accuratezza dato che vengono controllate tutte le parole, però non abbiamo comunque la certezza di trovare la parola esatta con questo metodo, infatti in parole dove cambia solamente un carattere è possibile che due o più opzioni siano ugualmente probabili.
Per esempio se la parola "casa" è diventata "casw" l'algoritmo dovrà scegliere tra le parole "casa" ,"caso" ,"case", "casi".

Un comportamento simile lo ha anche il confronto con parole che condividono almeno un 2-gram, infatti ci saranno così tante parole che condividono almeno un 2-gram che il guadagno in termini di tempo di questo filtro non è abbastanza. Usando la ricerca tra almeno un 3-gram e 4-gram invece abbiamo un notevole aumento di velocità mantenendo un buon hit-rate, ma comunque non abbastanza per risposte immediate.
Inoltre usare i 4-gram ha funzionato bene in questo caso, dato che delle parole scelte casualmente è facile che siano più lunghe del normale, se dovessimo correggere un documento con parole corte avremmo sicuramente un hit rate più basso.
Infine rimangono le ricerche su parole con un coefficiente di Jaccard superiore a 0.2 o 0.5.
Aumentando la dimensione degli n-gram usati e la soglia si può vedere come i tempi di esecuzione e gli hit rate scendano.

Il migliore compromesso che siamo in grado di trovare tra accuratezza e velocità sembra essere ricerca tra 3-gram con coefficiente di Jaccard superiore a 0.2, questo infatti ha un tempo medio di esecuzione minore di mezzo secondo e un hit rate del 63%.
Per applicazioni in cui è necessaria la massima accuratezza invece sembra più conveniente la ricerca tra 2-gram con coefficiente di Jaccard superiore a 0.2, questo con un tempo di poco più di un secondo offre un'accuratezza del 77%, che può essere considerata quasi ottima se confrontata con l'edit distance su tutti gli elementi del dizionario.