Edit Distance e N-grammi

Lorenzo Pesci

5 Luglio 2020

1 Introduzione

Vogliamo analizzare le differenze tra Edit Distance e N-grammi dato un lessico L e una stringa Q per trovare le parole in L più vicine a Q.

Entrambe sono tecniche che determinano quanto due stringhe siano simili.

Vengono utilizzate per la correzione di parole isolate, per correggere documenti, per suggerire query all'utente e per correzione di documenti antichi tramite OCR.

1.1 Edit Distance

La distanza di edit tra due stringhe A e B è il numero minimo di modifiche elementari che consentono di trasformare la A nella B. Per modifica elementare si intende:
- la cancellazione di un carattere
- la sostituzione di un carattere con un altro
- l'inserimento di un carattere
- la copia di un carattere
- lo scambio tra due caratteri vicini

1.2 N-grammi

Il problema con l'Edit Distance è il grande numero di confronti da effettuare con un dizionario.

Una possibile soluzione è utilizzare l'intersezione di n-grammi, ovvero enumerare tutti gli n-gram in Q e il L, usare un indice di n-gramma per trovare tutti i termini in L che contengono "abbastanza" n-grammi di Q.

La distanza tra due stringhe è calcolata tramite il **coefficiente di Jaccard**, definito come la dimensione dell'intersezione divisa per la dimensione dell'unione degli n-grammi ($J(A,B) = \frac{|A \cup B|}{|A \cap B|}$).

Nel nostro caso, A e B sono insiemi di n-grammi. Dalla definizione si può vedere che il coefficiente di Jaccard è compreso tra 0 e 1 ($0 < J(A,b) < 1$).

In particolare, il coefficiente è zero se A e B sono disgiunti ed è 1 se A e B hanno gli stessi elementi.

1.3 Considerazioni teoriche

Per come sono stati implementati, sia l'Edit Distance che gli N-grammi sono metodi che scorrono un'intero dizionario. La differenza sostanziale tra le due consiste nel numero di operazioni svolte per ogni parola: l'*edit_distance* esegue molte più operazioni di *ngram* poichè gli n-grammi sono stati calcolati a parte.

2 Analisi delle Operazioni di Edit Distance

Nella nostra trattazione useremo le seguenti notazioni:
- X e Y sono le stringhe da confrontare
- M e N sono le rispettive lunghezze
- c matrice dei costi
- op matrice delle operazioni
- i e j sono indici che vanno da 0 a m-1 e da 0 a n-1, utilizzati per scorrere matrici
- t tabella utilizzata per immagazzinare i costi delle operazioni ottime

2.1 edit distance(x, y)

La funzione *edit_distance* è un algoritmo che, prese in ingresso due stringhe, costruisce le matrici c $(M \times N)$ e *op* $(M \times N)$. C rappresenta la matrice di costi associata alle operazioni descritte nella matrice *op*, necessarie per convertire la stringa X nella stringa Y.

2.2 op sequence(op, i, j, t)

La funzione *op_sequence* è un algoritmo ricorsivo che, presa in ingresso la matrice delle operazioni (restituita da *edit_distance*), gli indici i e j (inzialmente m-1 e m-1) e la tabella t (inizialmente vuota), riempie t con le operazioni migliori da utilizzare nella trasformazione. La funzione restituisce il valore di ritorno della funzione *calc_cost*.

2.3 calc cost(t)

La funzione *calc_cost* è un algoritmo che, presa in ingresso la tabella t, restituisce un intero c che è la somma dei costi delle operazioni in t (trovate grazie a *op_sequence*).

3 Analisi delle Operazioni di N-Gram

Nella nostra trattazione useremo le seguenti notazioni:
- n è la lunghezza dell'n-gramma
- p è la parola da dividere in n-grammi

3.1 ngram(p, n)

La funzione *ngram* è un algoritmo che, presi in ingresso una parola p e la lunghezza dell'n-gramma n, restituisce un'array contenente gli n-grammi della parola data. (**Esempio**: *ngram*("algoritmi", 2) —> ['al', 'lg', 'go', 'or', 'ri', 'it', 'tm', 'mi']).

3.2 jaccard(ng1, ng2)

La funzione *jaccard* è un algoritmo che, presi in ingresso due array contenenti n-grammi qualsiasi (resitituiti da *ngram*), restituisce il coefficiente di Jaccard tra ng1 e ng2.

4 Descrizione Esperimenti e Documentazione del codice

Gli esperimenti sono stati svolti su un MacBook Pro con sistema operativo macOS Catalina, processore 2,2 GHz 6-Core Intel Core i7 e 16 GB di RAM.

Eventuali numeri casuali sono stati generati dalla funzione *random.randint()* importata dalla libreria *random*.

Il codice è articolato in otto file: *edit_distance.py*, *ngram.py*, *tests.py*, *exp.py*, *save_ngrams.py*, 60000*_parole_italiane.txt*, 2*_grams.txt*, 3*_grams.txt*, 4*_grams.txt*.

1) Nel file **edit_distance.py** è stato implementato l'algoritmo di Edit Distance per il calcolo della distanza tra due stringhe.

2) Nel file **ngram.py** è stata implementata la divisione di una parola in n-grammi e il calcolo del coefficiente di Jaccard dati due n-grammi.

3) Nel file **tests.py** sono stati implementati i seguenti esperimenti (dove non specificato, sono stati utilizzati bigrammi con coefficiente di Jaccard 0.6, e la soglia di Edit Distance è stata impostata a 3):

- Presa una parola a caso dal dizionario (60000*_parole_italiane.txt*), è stato misurato il tempo impiegato sia con Edit Distance che con N-grammi per trovarla.

- Prese 9 parole arbitrarie di lunghezza crescente (['a', 'ad', 'con', 'muro', 'zoppo', 'marito', 'sboccia', 'faticavo', 'abbattete']), è stato misurato il tempo di esecuzione e il numero di parole vicine trovate sia per Edit Distance che per N-grammi.

- Presa in input una parola, è stato misurato il numero di risultati trovati al variare della soglia di Edit Distance (1, 2, 3, 4, 5) e del coefficiente di Jaccard (0.5, 0.6, 0.7, 0.8, 0.9).

- Prese tre parole casuali di lunghezza ≥ 5, è stato misurato il numero di parole vicine trovate al variare del numero di n-grammi (2, 3, 4).

- Presa una parola in input, essa viene alterata (tramite la funzione $altera(word)$) e viene misurato il numero di risultati trovati al variare della soglia di Edit Distance (1, 2, 3, 4, 5) e del coefficiente di Jaccard (0.5, 0.6, 0.7, 0.8, 0.9). Inoltre si controlla anche se la parola originale è presente tra le parole vicine trovate.

4) Nel file **exp.py** sono stati eseguiti i test descritti nel file **tests.py**.

5) Nel file **save_ngrams.py** è stata implementata una funzione che a partire da una lista di 60000 parole (60000*_parole_italiane.txt*), genera un file $n\_grams.txt$ (con $2 \leq n \leq 4$) contenente la parola e l'n-gramma corrispondente.

5 Presentazione Dati Sperimentali

Vengono di seguito presentati i grafici relativi agli esperimenti svolti:

5.1 Tempo di esecuzione Edit Distance e N-grammi

Parola	**Tempo edit distance**	**Tempo n-gram**
elogiavate	3.032	0.543
continuato	2.185	0.356
ammazzero	0.578	0.112
smorzare	7.883	1.712
sequenza	6.624	1.403
assalti	0.748	0.229
ubertose	7.404	1.731
allegavamo	0.4385	0.1107
occupiate	4.907	0.966
espiasti	2.549	0.5325

Tabella 1: Tempo di esecuzione impiegato sia da Edit Distance che da N-grammi per trovare una parola casuale.

Per questo esperimento sono state prese 10 parole casuali dall'elenco di parole italiane (60000_*parole_italiane.txt*).

Per ognuna di esse è stato calcolato il tempo di esecuzione sia per Edit Distance che per N-grammi necessario per trovare la parola data.

Dai dati si può notare come l'algoritmo N-grammi sia sempre più rapido rispetto all'Edit Distance nella ricerca delle parole.Questo è dovuto al fatto che nella ricerca di parole N-gram esegue meno operazioni rispetto a edit-distance, la quale si rivela essere molto pesante per la ricerca di parole 'lontane' dall'inizio del dizionario.

5.2 Tempo di esecuzione di Edit Distance e N-grammi per parole di dimensione variabile

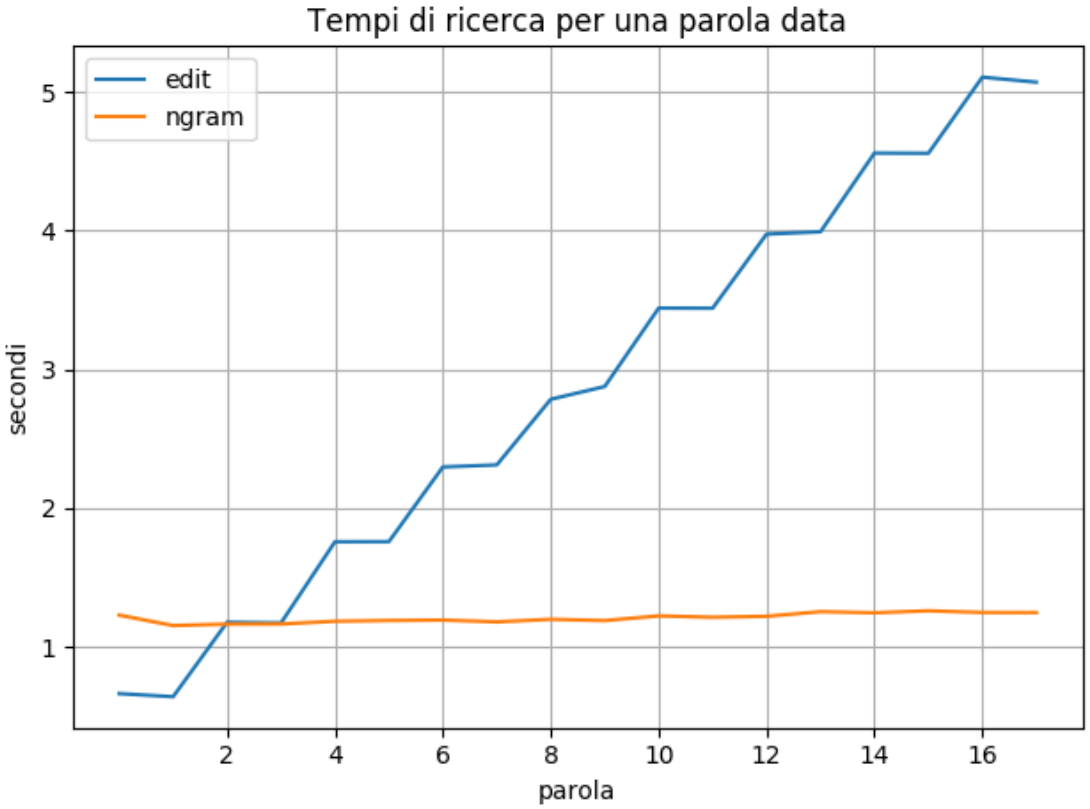

Figura 1: Tempo di esecuzione di Edit Distance e N-grammi per parole di dimensione variabile.

Per questo esperimento sono state prese 18 parole arbitrarie di lunghezza crescente dall'elenco di parole italiane (60000_*parole_italiane.txt*): 'a', 'c', 'ad', 'ed', 'con', 'dal', 'muro', 'cane', 'zoppo', 'gatto', 'marito', 'pulito', 'sboccia', 'cantare', 'faticavo', 'fratelli', 'penzolano', 'cameriere'.

Per ognuna delle parole scelte è stato applicato sia l'algoritmo di Edit Distance che N-grammi.

Dal grafico si può notare che il tempo dell'algoritmo N-grammi è quasi costante ($\sim 3 secondi$) qualsiasi sia la lunghezza della parola.

I tempi dell'Edit Distance, al contrario, crescono in maniera lineare all'aumentare della lunghezza della parola scelta.

In conclusione, se si ha a che fare con parole lunghe, l'algoritmo di N-grammi è nettamente più efficiente.

5.3 Numero di parole vicine trovate con Edit Distance e N-grammi per parole di dimensione variabile

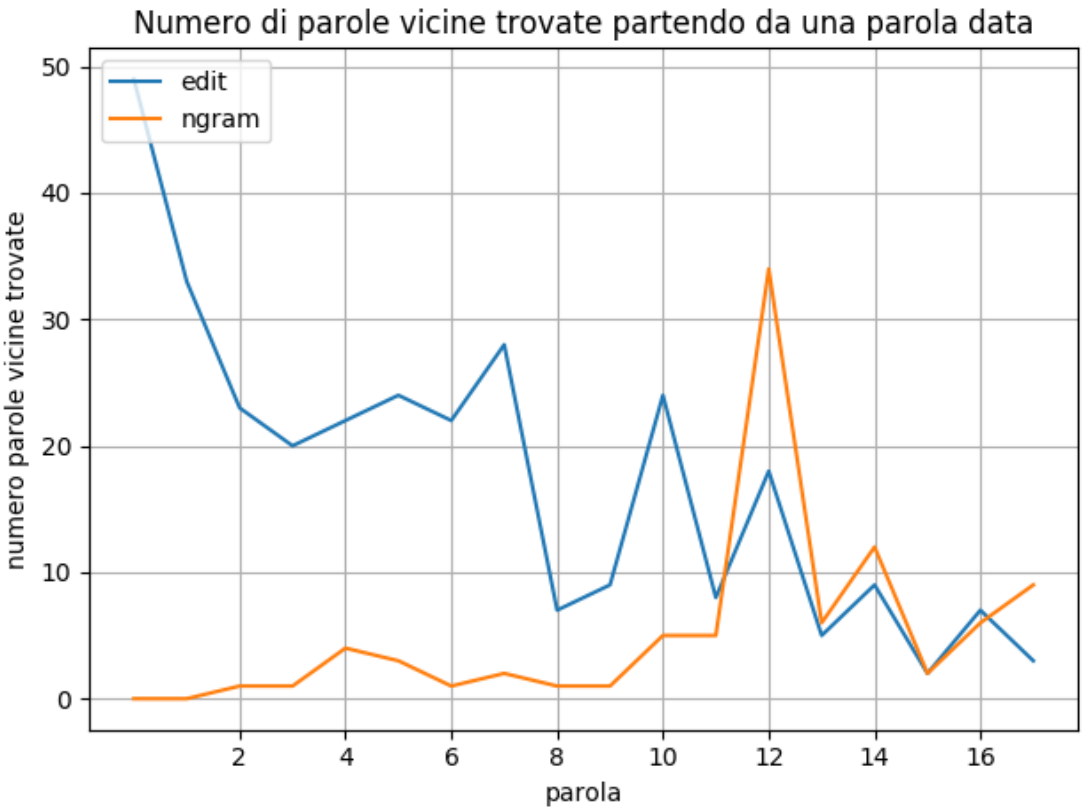

Figura 2: Numero di parole vicine trovate con Edit Distance e N-grammi per parole di dimensione variabile.

Anche per questo esperimento sono state prese 18 parole arbitrarie di lunghezza crescente dall'elenco di parole italiane (60000_*parole_italiane.txt*): 'a', 'c', 'ad', 'ed', 'con', 'dal', 'muro', 'cane', 'zoppo', 'gatto', 'marito', 'pulito', 'sboccia', 'cantare', 'faticavo', 'fratelli', 'penzolano', 'cameriere'.

Per ognuna delle parole scelte è stato applicato sia l'algoritmo di Edit Distance che N-grammi.

Dal grafico si può notare che il numero di parole vicine trovate dell'algoritmo N-grammi è crescente, con il crescere della lunghezza della parola.

Il numero di parole trovate da Edit Distance, al contrario, decrescono all'aumentare della lunghezza della parola scelta.

5.4 Numero di risultati trovati al variare della soglia di *Edit distance*

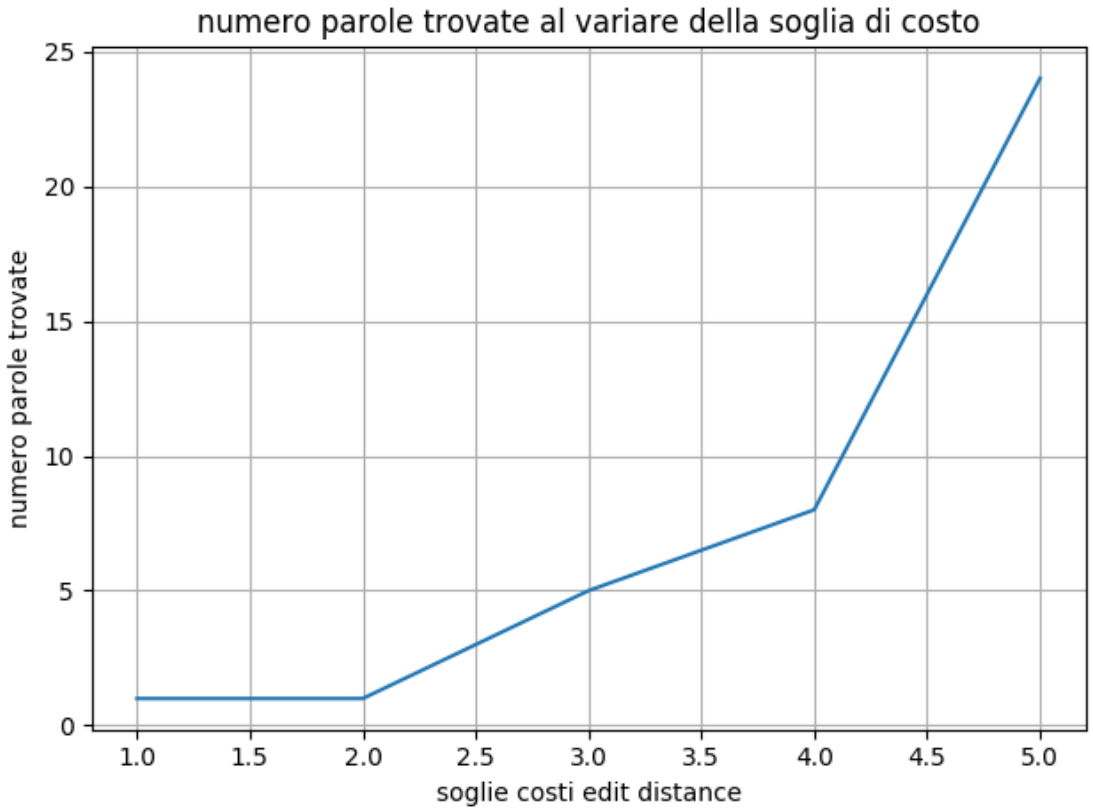

Figura 3: Numero di parole vicine ad una parola data ('promosso') al variare della soglia di *Edit distance*

Per questo esperimento è stata presa in ingresso una parola chiesta all'utente, in questo caso 'promosso', alla quale è stato applicato l'algoritmo *Edit distance* più volte.

Per ogni volta che viene applicato *Edit distance* la soglia di errore viene aumentata di 1, per osservare come varia l'insieme di parole restituite dall'algoritmo.

Dal grafico si può notare che il numero di parole vicine trovate dell'algoritmo *Edit distance* sia proporzionale all'aumento della soglia di *Edit distance*.

5.5 Numero di risultati trovati al variare del coefficiente di Jaccard

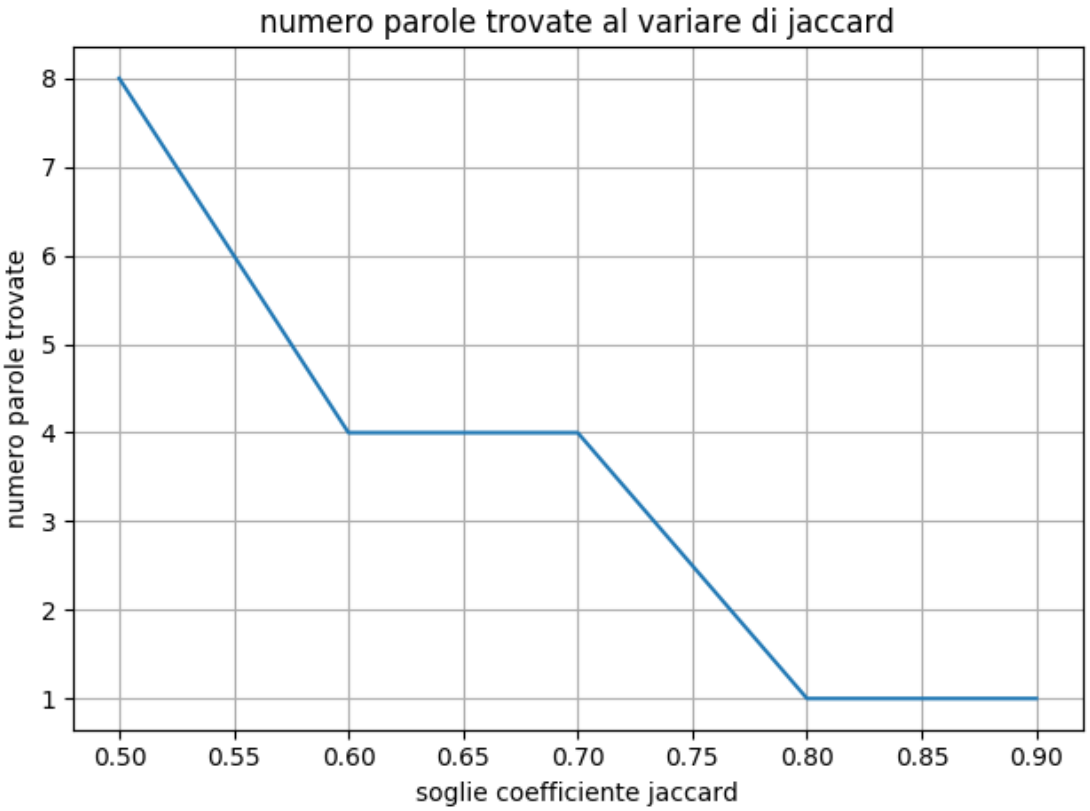

Figura 4: Numero di parole vicine ad una parola data ('promosso',) al variare della soglia di *Edit distance*

Anche per questo esperimento è stata presa in ingresso una parola chiesta all'utente, in questo caso 'promosso', alla quale è stato applicato l'algoritmo $N-gram$ più volte.

Per ogni volta che viene applicato $N-gram$ il coefficiente di Jaccard vine aumentato di 0.1, per osservare come varia l'insieme di parole restituite dall'algoritmo.

Dal grafico si può notare che il numero di parole vicine trovate dell'algoritmo $N-gram$ sia inversamente proporzionale all'aumento del coefficiente di Jaccard.

5.6 Numero di parole vicine trovate al variare del numero di N-grammi

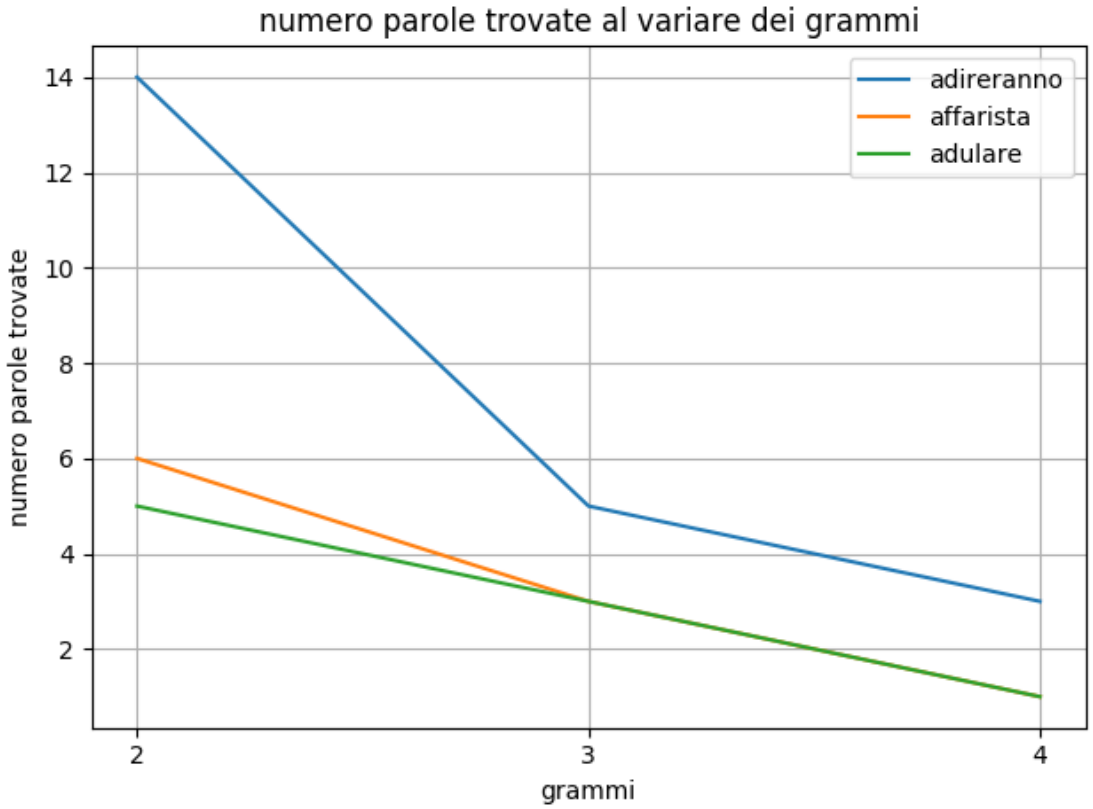

Figura 5: Numero di parole vicine a parole casuali trovate utilizzando bi-grammi, trigrammi e quadrigrammi

Per questo esperimento sono state utilizzate tre parole casuali ('adireranno', 'affarista', 'adulare'), alle quali è stato applicato l'algoritmo $N - gram$ più volte.

Per ogni volta che viene applicato $N - gram$ si utilizza diversi valori di n, in particolare [2, 3, 4].

Dal grafico, e da molteplici esperimenti eseguiti, si nota come il numero di parole trovate non dipenda dal valore di n, ma varia ogni volta a seconda della parola scelta e della sua lunghezza.
z

5.7 Risultati della ricerca di una parola alterata tramite Edit distance e N-gram al variare della soglia e del coefficiente di jaccard.

Per questo esperimento è stata presa in ingresso una parola chiesta all'utente, in questo caso 'ventisette' sulla quale è stata eseguita un'alterazione casuale tra le tre possibili :
-Scambio di due caratteri casuali
-Eliminazione di un carattere casuale
-Aggiunta di un carattere casuale

In questo caso la parola 'ventisette' è stata alterata tramite uno scambia di caratteri in 'ventisetet'.

Soglia di costo	**Parola originale trovata**	**Parole vicine trovate**
1	no	0
2	si	1
3	si	1
4	si	2
5	si	7

Tabella 2: Tabella che mostra i risultati dell'esperimento al variare della soglia di costo di *Edit distance*.

Coefficiente di Jaccard	**Parola originale trovata**	**Parole vicine trovate**
0.5	si	3
0.6	si	2
0.7	si	1
0.8	si	1
0.9	si	1

Tabella 3: Tabella che mostra i risultati dell'esperimento al variare del coefficiente di Jaccard.

Dalle tabelle, e da molteplici esperimenti eseguiti, si nota che generalmente la parola originale è ritrovata più volte tramite *Edit distance*, rispetto ad $N - gram$, anche con soglie di errore alte.

Questo fatto però non è sempre vero, infatti la possibilità di ritrovare la parola originaria, sia con *Edit distance* sia con $N - gram$, dipende dalla parola scelta, dal tipo di alterazione e dalla parte specifica di parola che viene alterata.

6 Conclusioni

In conclusione, come ci si aspettava dalla teoria, gli esperimenti hanno evidenziato differenze tra Edit Distance e N-grammi.

Infatti per quanto riguarda i **tempi di esecuzione**, Edit Distance si è effettivamente rivelato più lento rispetto agli N-grammi in quanto deve eseguire molti più confronti.
Questo infatti è evidenziato soprattutto nella ricerca di parole 'lontane' dall'inizio del vocabolario, per le quali il tempo di ricerca di Edit distance supera di gran lunga quello di N-gram per la stessa parola, proprio per la gran differenza di operazioni eseguite sulle singole parole.
Per quanto riguarda il **numero di parole vicine trovate**, la situazione è capovolta: Edit distance riesce, generalmente, a trovare molte parole vicine ad una parola data, anche con soglie di errore alte, mentre N-gram trova quasi sempre un numero minore di parole vicine.
Questa differenza è dovuta sopratutto al coefficiente dei costi sul quale si basa Edit distance, nel nostro caso il costo della copia di un caratere era nulla, e ciò permette ad Edit distance di trovare le parore molto simili ad una data anche soglie di errore alta.
Dagli esperimenti svolti è possibile ritenere che la funzione Edit distance sia da preferirsi ad N-gram per la realizzazione di un correttore ortografico, poichè generalmente riesce a ritrovare più facilmente la parola originale.