

# UNIVERSITÀ DEGLI STUDI DI FIRENZE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE



Corso di Laurea in Ingegneria Informatica

## Edit Distance

Professore:

**Ch.mo Prof.  
Simone Marinai**

Candidato:

**Hamidreza Hashemi  
Mat. 5996260**

Corso:

**Algoritmi e Strutture Dati**

ANNO ACCADEMICO 2020/2021

*Dedicata a coloro che  
ci provano ogni giorno.*

---

# INDICE

<b>Introduzione</b>	<b>3</b>
<b>1 Edit Distance</b>	<b>4</b>
<b>2 Implementazione</b>	<b>6</b>
<b>3 Conclusione e Confronto</b>	<b>8</b>

---

# INTRODUZIONE

In questo esercizio vediamo insieme l'algoritmo edit distance e l'implementazione nel linguaggio python. Questo algoritmo serve per trovare la parola più simile ad una query in un lessico di parole.

---

# CAPITOLO 1

---

## EDIT DISTANCE

L'algoritmo Edit Distance trova la distanza tra una parola e un'altra. La distanza qui vuol dire il numero di volte di fare certe operazioni per arrivare alla parola di lessico. Questa distanza si chiama la distanza di Levenshtein. Queste operazioni sono :

- Inserimento
- Cancellazione
- Sostituzione

```
def editDistDP(str1, str2, m, n):  
    cost = np.zeros((m + 1, n + 1))  
    for i in range(m + 1):  
        for j in range(n + 1):  
            if i == 0:  
                cost[i, j] = j  
            elif j == 0:  
                cost[i, j] = i  
            elif str1[i - 1] == str2[j - 1]:  
                cost[i, j] = cost[i - 1, j - 1]  
            else:  
                cost[i, j] = 1 + min(cost[i, j - 1], cost[i - 1, j], cost[i - 1, j - 1])  
    return cost[m, n]
```

Figura 1.1: Edit distance in python

# 1. EDIT DISTANCE

		Empty String	String "B"								
		""	A	L	T	R	U	I	S	M	
Empty String	""	0	1	2	3	4	5	6	7	8	
	P	1	1	2	3	4	5	6	7	8	
String "A"	L	2	2	1	2	3	4	5	6	7	
	A	3	2	2	2	3	4	5	6	7	
	S	4	3	3	3	3	4	5	5	6	
	M	5	4	4	4	4	4	5	6	5	
	A	6	5	5	5	5	5	5	6	6	

Figura 1.2: Esempio di Edit Distance

Il costo temporale di edit distance è  $\theta(m \cdot n)$  dove m e n sono le lunghezze di stringhe. Questo algoritmo può essere applicato in Biologia ed elaborazione del linguaggio naturale.

---

---

## CAPITOLO 2

---

### IMPLEMENTAZIONE

Ci sono diversi modi per implementare algoritmo edit distance. Uno di questi è l'implementazione con N-Grams. N-Grams divide una stringa e crea una sottosequenza di n elementi. Questa implementazione normalmente serve per migliorare il costo temporale dell'algoritmo perchè prima di trovare la distanza tra le due parole lunghe, va a vedere se esiste una sottosequenza di N-Gram nella parola di lessico e poi va a calcolare la distanza. Vediamo un'esempio qui sotto :

..AGCTTCGA..  $\longrightarrow$  .., AGC, GCT, CTT, TTC, TCG, CGA,..

Questo è un esempio di **3-Gram**.

Si può trovare la distanza tra le due parole calcolando il **Coefficiente di Jaccard** tra gli N-Grams. la formula del coefficiente di Jaccard è :

$$JC = \frac{A \cap B}{A \cup B}, \text{ dove A e B sono i set di N-Grams}$$

---

## 2. IMPLEMENTAZIONE

---

```
def run(self):
    ngramstrings = []
    for element in self.query:
        ngramstrings.append(letters(element))
    dojaccardcalculations = False
    print("Starting " + self.name)
    for line in self.linesarray:
        # Vede se esiste un ngram nella parola di lessico. Altrimenti non fa i calcoli.
        if any(word in line for word in ngramstrings):
            line_set = set(ngrams(line, self.ngram))
            # Calcola il coefficiente jaccard tra query e una parola
            jc = jaccard_coef(self.query, line_set)
            if jc > self.top3[0]:
                self.top3[0] = jc
                self.similar_words[0] = line
            elif jc > self.top3[1]:
                self.top3[1] = jc
                self.similar_words[1] = line
            elif jc > self.top3[2]:
                self.top3[2] = jc
                self.similar_words[2] = line
    print("Exiting " + self.name)
```

Figura 2.1: ngrams in thread

In questo esercizio è stato implementato Edit distance semplice e con gli N-Grams. Il programma trova le parole che hanno almeno una sottosequenza di N-Grams nella parola di query. Poi va a calcolare il coefficiente di Jaccard tra gli N-Grams e da come output i primi 3 valori più simili rispetto alla parola query. Dopo calcola la distanza tra queste 3 parole con la parola query. Questo serve per ottimizzare il costo temporale. Come vediamo nella figura di sopra, il calcolatore di jaccard è stato implementato come un Thread di python. Ogni thread può cercare la parola più simile in un lessico.



---

---

## CAPITOLO 3

---

### CONCLUSIONE E CONFRONTO

Abbiamo provato diversi algoritmi per trovare la somiglianza tra 2 parole. Come un esempio è stato provato la somiglianza tra la parola "tigr" e un lessico di parole italiane con l'algoritmo di 3grams:

```
Starting th1
Exiting th1
3Grams !!
Word : tигра
with Jaccard Coefficient : — 0.67 —
has the minimum distance : 1.0
Word : tigre
with Jaccard Coefficient : — 0.67 —
has the distance : 1.0
Word : tigri
with Jaccard Coefficient : — 0.67 —
has the distance : 1.0
Spent time : 0.66
```

Tabella 3.1: Output generato

---

### 3. CONCLUSIONE E CONFRONTO

---

provando da 1-Gram e aumentando la divisione della query si capisce che il tempo diminuisce e la precisione aumenta. Vediamo dalla tabella 3.2, il risultato di una serie di esperimentazioni.

Tempo trascorso in sec.					
Query	1-Gram	2-Grams	3-Grams	4-Grams	Edit Distance
<b>can</b>	5.3722	2.1627	0.6163	err	67.3821
<b>tigr</b>	5.5277	2.0518	0.6595	0.5442	77.3288
<b>conigl</b>	5.5500	2.1150	1.0300	0.6344	112.7757

Tabella 3.2: confronto di tempo tra N-Grams.

Come vediamo per la query "can" abbiamo un errore in 4-grams perchè non è divisibile a 4 caratteri. Dalla tabella 3.3 notiamo che aumentando la divisione abbiamo le parole più precise :

Parole trovate					
Query	1-Gram	2-Grams	3-Grams	4-Grams	Edit Distance
<b>can</b>	accana anca can	can cana cancan	can cana cane	err	can
<b>tigr</b>	tigri aggirati argati	tigra tigre tigri	tigra tigre tigri	tigra tigre tigri	tigi
<b>conigl</b>	ciglioni cigolino cinciglio	conigli coniglia coniglie	conigli coniglia coniglie	conigli coniglia coniglie	conigli

Tabella 3.3: confronto di parole tra N-Grams.

### 3. CONCLUSIONE E CONFRONTO

---

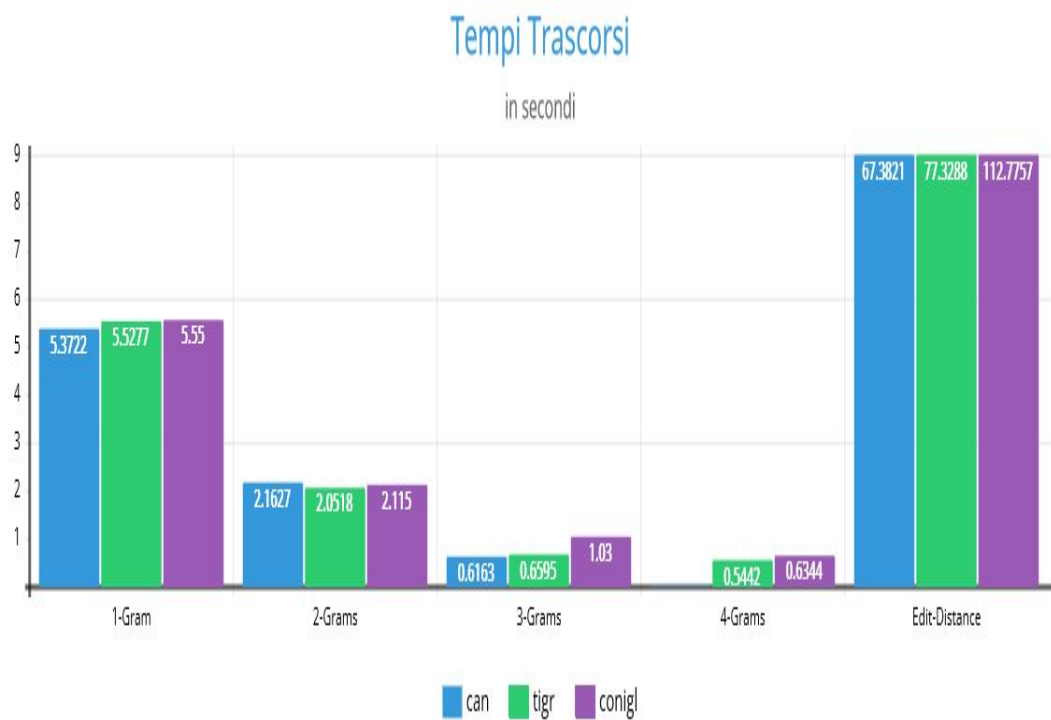


Figura 3.1: tempi trascorsi in sec.

Possiamo notare che L'algoritmo Edit Distance è notevolmente più costoso rispetto ai N-Grams

\*\*\*