L'esercitazione prevede l'implementazione, ispirandosi al Text Tiling, di un algoritmo di segmentazione del testo:

1. Usando informazioni come frequenze (globali, locali), co-occorrenze, risorse semantiche (WordNet, etc.), applicando step di preprocessing (as usual), etc.
    - La scelta del testo è a discrezione dello studente.

4.2 Svolgimento

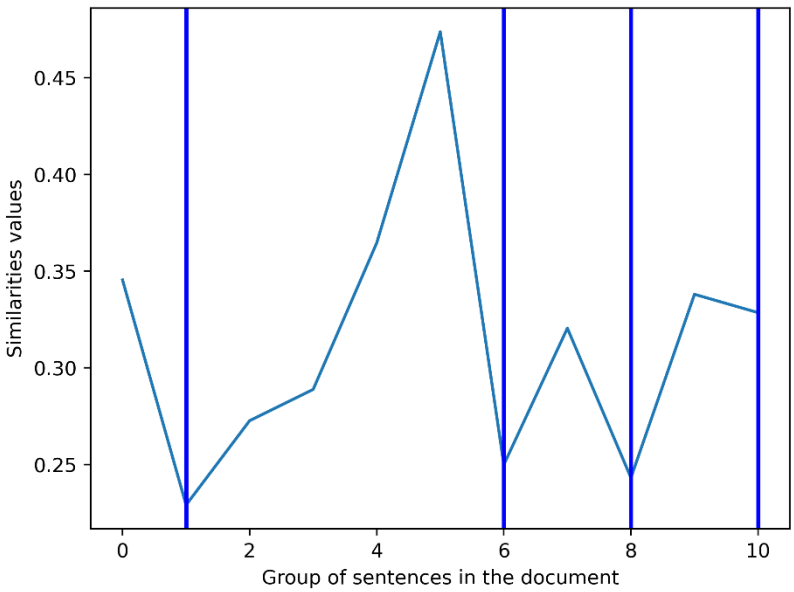
Il metodo del text tiling ha lo scopo di individuare le parti con significato simile all'interno del documento per poi segmentarlo. Inizialmente il testo viene separato in finestre di lunghezza fissa. Quindi il documento da analizzare (*historyOfRome.txt*) viene letto come una lista di frasi (83 per la precisione), grazie alla funzione `getSentences()`, che usa la funzione nativa di NLTK *tokenize,* per suddividere il documento in frasi. Il primo passo dunque è quello di effettuare una segmentazione del testo in finestre della stessa dimensione (nel mio caso pari a 6 frasi l'una), con la funzione `divideDocument()`, che restituisce una lista di liste di frasi (che serviranno per l'output) e una lista di liste di parole, ossia le frasi preprocessate (tokenizzazione, rimozione stopwords e punteggiatura, lemmatizzazione), che serviranno poi per calcolare la coesione tra finestre.

La coesione tra le finestre viene calcolata in base al numero di termini che co-occorrono nelle due finestre.
L'idea dell'algoritmo è quella di calcolare la coesione di una finestra rispetto a quella successiva, sulla base dei termini rilevanti presenti in ogni finestra, attraverso la funzione `getSimilarity()`:

```
def getSimilarity(sentences, sentences1):
    count = Counter(sentences)
    count1 = Counter(sentences1)
    word = [*count]
    word1 = [*count1]
    sim = 0
    for w in word:
        freq = count[w]
        if w in word1:
            freq += count1[w]
        else: freq = 0
        sim += freq

    return sim/len(sentences)
```

La funzione utilizza il *Counter* di python per contare le occorrenze di ogni parola e poi se la parola presente in un paragrafo è presente anche nell'altro, viene aggiunta la somma delle loro occorrenze alla variabile *sim* che verrà poi normalizzata e restituita dalla funzione.
Il risultato dopo aver esaminato tutte le finestre sarà una lista di similarità tra paragrafi. Per i principi del text tiling i migliori breaking points sono i "minimi", ossia valori più bassi circondati da valori più alti. Attraverso la funzione `getLocalMin()` recupero appunti i minimi locali all'interno della sequenza di similarità appena calcolata, ottenendo i seguenti split points:



Il testo quindi viene diviso utilizzando questi nuovi breaking points e il risultato è abbastanza soddisfacente, come si evince dall'output, poiché il documento in questione (La storia di Roma) viene diviso quasi perfettamente per dinastie e periodi storici che hanno contraddistinto l'ascesa e la discesa dell'Impero Romano.