Relazione Esercizio 4

In questa esercitazione, abbiamo scritto ispirandoci al Text Tiling un algoritmo di Document Segmentation. I passi iniziali di tale algoritmo sono molto semplici, poiché abbiamo voluto rimanere il più possibile fedeli all'implementazione del Text Tilling di NLTK. L'algoritmo si comporta nel seguente modo:

1. Importato e inizializzato nasari all'interno di un dizionario. In una stuttura di configurazione è possibile impostare gli n primi elementi da salvare in dizionario per ogni word.

2. Tramite NLTK abbiamo mappato il corpus in una lista di sentences e generato due strutture. una con le sentences come lista di stringhe e una con le sentences come lista di lista di tokens.

3. Per ogni token (parola) in ogni frase è stata calcolata la similarietà tra quest'ultimo e tutti gli altri token prima nella frase precedente e poi in quella successiva. Per calcolare tale similarietà si è ricorsi prima all'uso della metrica del Weighted Overlap per computare la similarietà tra i due vettori Nasari dei due token,

$$WO(v_1, v_2) = \frac{\sum_{q \in O} \left(rank(q, v_1) + rank(q, v_2)\right)^{-1}}{\sum_{i=1}^{|O|} (2i)^{-1}}$$

   poi successivamente, si è fatto ricorso allo Square Root Weighhted Overlap per calcolare la similarietà finale delle due parole a partire dal Weighted Overlap calcolato in precedenza. Infine, la similarietà di ogni riga sarà equivalente alla somma della similarietà della riga corrente e della precedente con la similarietà della riga corrente con la successiva.

$$sim(w_1, w_2) = \max_{v_1 \in \mathcal{C}_{w_1}, v_2 \in \mathcal{C}_{w_2}} \sqrt{WO(v_1, v_2)}$$

4. Essendo che recedentemente abbiamo ottenuto un vettore di similarità assegnando uno score ad ogni frase. Per cui è stato usato K-Means per clusterizzare gli scores e individuare il numero minimo di cluster che definiranno le finistre iniziali per la segmentazione. Il miglior valore di cluster è stato calcolato automaticamente con la Silouhette Analysis

> **Silhouette analysis**:
>
> Another more automated approach would be to build a collection of k-means clustering models with a range of values for k and then evaluate each model to determine the optimal number of clusters.
>
> We can use Silhouette analysis to evaluate each model. A Silhouette coefficient is calculated for observation, which is then averaged to determine the Silhouette score. **The coefficient combines the average within-cluster distance with average nearest-cluster distance to assign a value between -1 and 1**. A value below zero denotes that the observation is probably in the wrong cluster and a value closer to 1 denotes that the observation is a great fit for the cluster and clearly separated from other clusters. This coefficient essentially measures how close an

> observation is to neighboring clusters, where it is desirable to be the maximum distance possible from neighboring clusters.
>
> We can automatically determine the best number of clusters, k, by selecting the model which yields the highest Silhouette score.

```
print("\tComputing clusters using K-Means...")
sentences_similarities = np.array(similarities)
data = sentences_similarities.reshape(-1, 1)  # needed for cluster
computin

# Sets the best cluster size within the minimum and maximum supplied.
Eg.: 2,10
clusters_size_ranges = np.arange(2, 10)
clusters_sizes = {}
for size in clusters_size_ranges:
    model = KMeans(n_clusters=size).fit(data)
    predictions = model.predict(data)
    clusters_sizes[size] = silhouette_score(data, predictions)
best_clusters_size = max(clusters_sizes, key=clusters_sizes.get)
print("\t\tThe best cluster group size is:
{}".format(best_clusters_size))
```

Note: è stato dato il range iniziale (2, 10) in cui generare modelli per valutre cluster. Sviluppi futuri potrebbero prevedere l'idea di ricalcolare con un range più alto laddove la *clusters size* ottimale sia maggiore di un paramentro (Es. 70% del range).

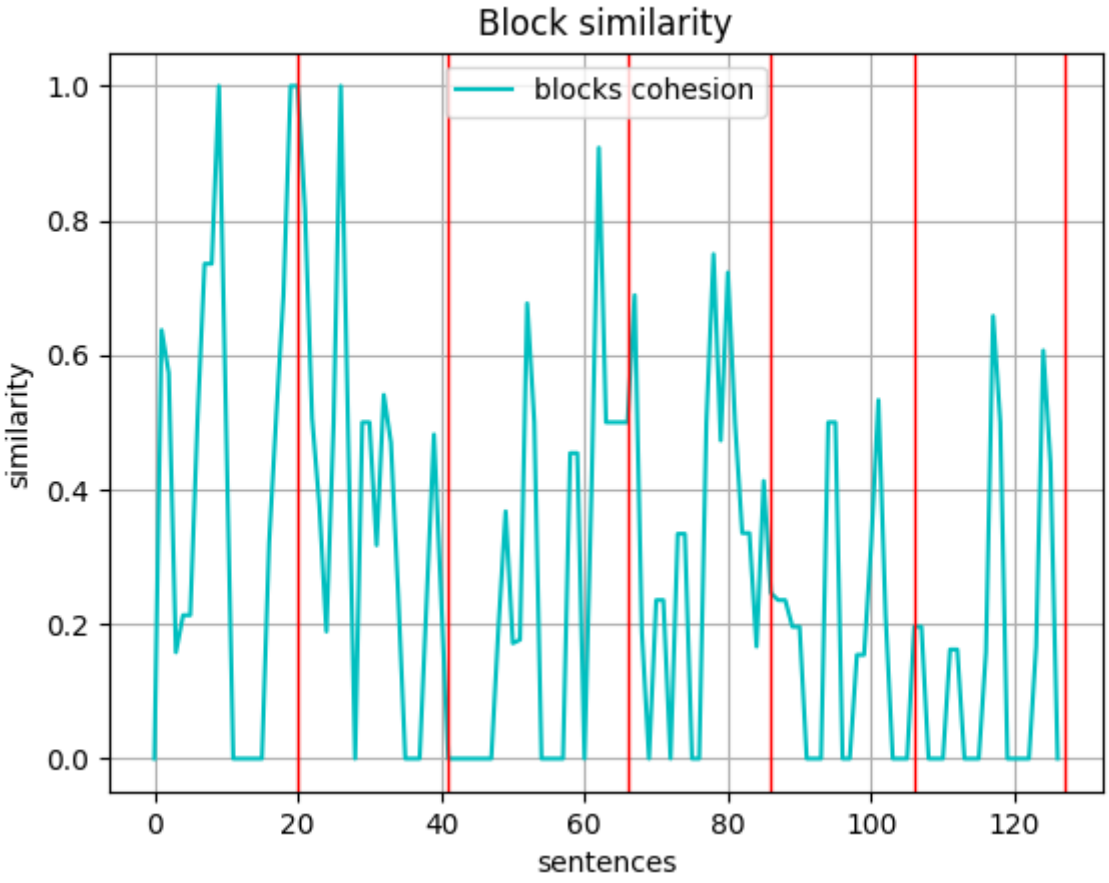
5. A questo punto il numero minimo ottimale di clusters è noto. In un primo momento abbiamo provato a rieseguire Kmeans per verificare la distribuzione dei cluster sul corpus. Non riuscendo a trovare un algoritmo efficiente per i casi in cui i cluster si ripetono più volte abbiamo optato per il semplice shift delle finestre rispetto alla similarità delle frasi.

6. Il corpus viene suddiviso in un numero di breakpoints uguale al numero di clusters calcolati. per ognuno di queti break point si verifica la similarità delle frasi di frontiera con le finestre adiacenti ed eventualmente si sposta il break point.

7. L'algorimo termina quando non ci sono più breakpoints da spostare

Conclusioni

A fine esecuzione, l'algoritmo descritto, ha dato in output il seguente plot:



come si può notare, K-Means a fine iterazioni è stato in grado basandosi sulal simiarietà dellew frasi di posizionare in modo *"aprossimativamente corretto"* i giusti breakpoint. Un possibile futuro sviluppo futuro consisterebbe nel verificare la similarietà semantica e non quella lessicale come nel nostro algoritmo.

Osservazioni aggiuntive

- A seguito dei nostri esperimenti sul testo `snowden.txt` (Edward Snowden: the whistleblower behind the NSA surveillance revelations), abbiamo **ipotizzato** che i testi autobiografici presentino una matrice di similarietà e dei cluster molto sparse.

Appendice

Matrice delle similarietà

```
array([0.        , 0.70412415, 0.20412415, 0.41493811, 0.57305199,
       0.15811388, 0.21320072, 0.21320072, 0.        , 0.        ,
       0.23570226, 0.73570226, 0.        , 0.5       , 0.        ,
       0.        , 0.5       , 0.81622777, 0.31622777, 0.23570226,
       0.5029635 , 0.45624348, 0.68898224, 0.65811388, 0.15811388,
       0.5       , 0.5       , 0.31725282, 0.56725282, 0.57203059,
       0.32203059, 0.5       , 0.        , 0.72237479, 0.22237479,
       0.31725282, 0.54085962, 0.2236068 , 0.17149859, 0.52505198,
       0.35355339, 0.        , 0.2409996 , 0.4819992 , 0.66225829,
       0.42125869, 0.        , 0.        , 0.19611614, 0.36761472,
       0.17149859, 0.        , 0.20412415, 0.70412415, 0.5       ,
```

```
       0.        , 0.        , 0.        , 0.43952454, 0.59028021,
       0.55900396, 0.90824829, 0.        , 0.        , 0.5       ,
       0.5       , 0.5       , 0.16666667, 0.40236893, 0.23570226,
       0.        , 0.33419801, 0.33419801, 0.        , 0.31622777,
       0.81622777, 0.75      , 0.47301681, 0.72277262, 0.49975581,
       0.31622777, 0.65156306, 0.3353353 , 0.25      , 0.49618298,
       0.24618298, 0.23570226, 0.48188524, 0.24618298, 0.19611614,
       0.19611614, 0.26726124, 0.26726124, 0.        , 0.36324158,
       0.36324158, 0.31990258, 0.31990258, 0.21320072, 0.40218295,
       0.18898224, 0.20412415, 0.40024028, 0.19611614, 0.        ,
       0.        , 0.16222142, 0.32444284, 0.42948266, 0.26726124,
       0.        , 0.        , 0.5       , 0.65430335, 0.3086067 ,
       0.15430335, 0.        , 0.28867513, 0.72876374, 0.        ])
```

Matrice dei clusters

```
array([2, 4, 1, 3, 0, 1, 1, 1, 2, 2, 1, 4, 2, 0, 2, 2, 0, 4, 3, 1, 0, 0,
       4, 4, 1, 0, 0, 3, 0, 0, 3, 0, 2, 4, 1, 3, 0, 1, 1, 0, 3, 2, 1, 0,
       4, 3, 2, 2, 1, 3, 1, 2, 1, 4, 0, 2, 2, 2, 0, 0, 0, 4, 2, 2, 0, 0,
       0, 1, 3, 1, 2, 3, 3, 2, 3, 4, 4, 0, 4, 0, 3, 4, 3, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 2, 3, 3, 3, 3, 1, 3, 1, 1, 3, 1, 2, 2, 1, 3, 0, 1,
       2, 2, 0, 4, 3, 1, 2, 3, 4, 2], dtype=int32)
```