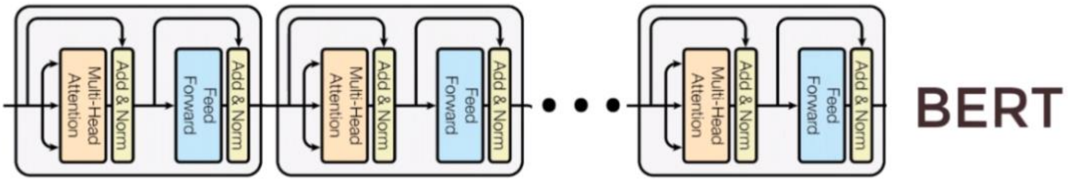
Bert

A bert model is a deep learning architecture based on a multi-head attention mechanism that weights the importance of each part of the input data differently. It follows an encoder-decoder structure but does not rely on recurrence and convolutions in order to generate an output. Transformers are designed to process sequential input data.



Process

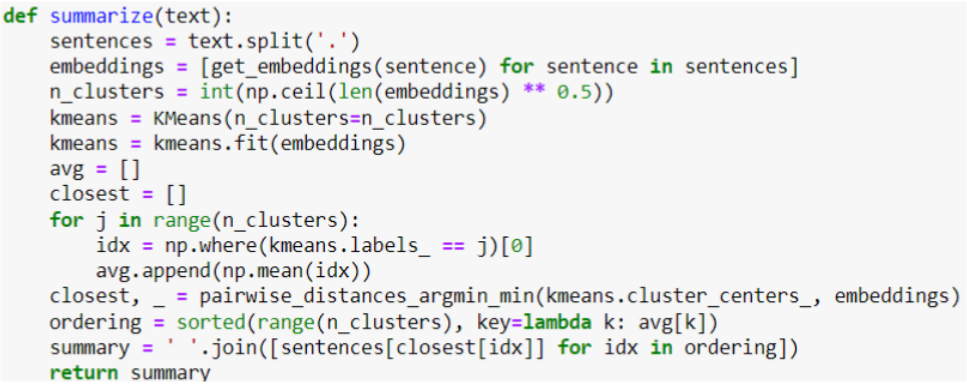
- **BERT Embeddings:** **The input text is tokenized and passed through a pre-trained BERT model to obtain contextual embeddings for each word/token in the text. These embeddings capture the semantic meaning and context of each word within the sentence.**

- **Sentence Embeddings:** **The contextual embeddings obtained from BERT are aggregated to represent each sentence in the input text. Various aggregation methods can be used, such as averaging or pooling the embeddings of individual tokens within the sentence.**

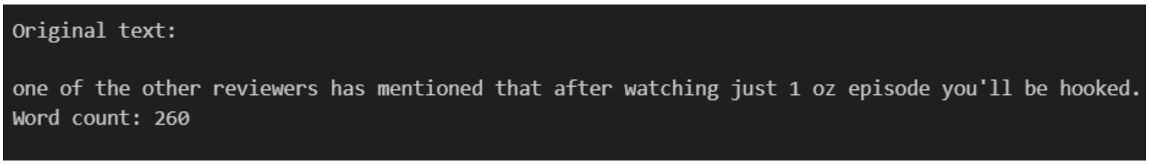
- **Clustering with K-Means:** **The sentence embeddings are then clustered using the K-Means algorithm. K-Means partitions the sentence embeddings into K clusters based on their similarity, where K is the desired number of sentences in the summary.**

- **Sentence Selection:** **Once the clusters are formed, the centroid of each cluster is computed. The sentences closest to each centroid are selected as representatives of their respective clusters and included in the summary.**

- **Summary Generation:** **The selected sentences from each cluster are concatenated to form the final summary.**

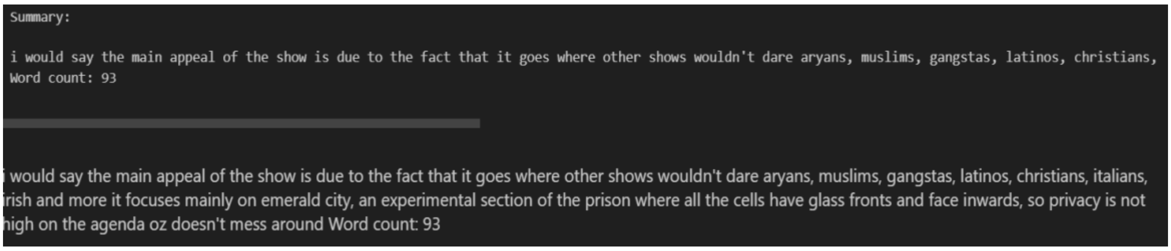


```python
def summarize(text):
    sentences = text.split('.')
    embeddings = [get_embeddings(sentence) for sentence in sentences]
    n_clusters = int(np.ceil(len(embeddings) ** 0.5))
    kmeans = KMeans(n_clusters=n_clusters)
    kmeans = kmeans.fit(embeddings)
    avg = []
    closest = []
    for j in range(n_clusters):
        idx = np.where(kmeans.labels_ == j)[0]
        avg.append(np.mean(idx))
    closest, _ = pairwise_distances_argmin_min(kmeans.cluster_centers_, embeddings)
    ordering = sorted(range(n_clusters), key=lambda k: avg[k])
    summary = ' '.join([sentences[closest[idx]] for idx in ordering])
    return summary
```



```
Original text:

one of the other reviewers has mentioned that after watching just 1 oz episode you'll be hooked.
Word count: 260
```



```
Summary:

i would say the main appeal of the show is due to the fact that it goes where other shows wouldn't dare aryans, muslims, gangstas, latinos, christians,
Word count: 93
```

i would say the main appeal of the show is due to the fact that it goes where other shows wouldn't dare aryans, muslims, gangstas, latinos, christians, italians, irish and more it focuses mainly on emerald city, an experimental section of the prison where all the cells have glass fronts and face inwards, so privacy is not high on the agenda oz doesn't mess around Word count: 93

Conclusion

In this study, we employed sentence embedding to extract meaningful clusters from a dataset. We utilized the K-means algorithm to partition the data into clusters, with the number of clusters determined by the square root of the total number of embeddings. Each cluster was then iteratively analyzed to assign index values closest to the centroid, facilitating the calculation of the average index values for the order of the given cluster, thus establishing final centroids. Through this process, we successfully extracted key points that converged to summarize the dataset. This method offers a systematic approach to distill large volumes of text data into concise and representative summaries, enabling efficient comprehension and analysis