**Final Project – Natural Language Processing – Omer Blau and Ofek Cohen**

**Data Processing**

We initially implemented advanced text processing with a cleaning function that expanded contractions (using the contractions library) and removed special characters, tokenization, stop word removal, and lemmatization. This version significantly amplified the differences between the semantics of the sentences, but our sentences were different than the sentences in the solution file, so we couldn’t use the evaluation function.  
The final function incorporates configurable parameters for each processing step. This flexible design enabled us to experiment with various cleaning configurations without code modifications. Those are the final params for the cleaning-text function:  
 params = { **"expand contractions": True, "remove special": True**, "tokenize": False, "remove stop words": False, "lemmatize": False }

To generate embeddings, we used the "SentenceTransformer" (all-MiniLM-L6-v2) model that encodes sentences into vector representations. Additionally, the embeddings were **normalized** to improve the clustering process.

**Clustering**

We followed the clustering algorithm suggested in the project assignment.  
The clustering algorithm begins without any predefined clusters, each request is initially unassigned. As the algorithm iterates through the dataset, it attempts to assign each request to an existing cluster if the euclidean distance to the cluster's centroid is within a given threshold (We chose 0.87). If a request does not fit into any existing cluster, a new cluster is created, and the request's embedding is the centroid of the new cluster. Centroids are recalculated after each iteration by taking the mean of the embeddings in each cluster. The algorithm iterates through the requests multiple times to reassign them if a closer cluster is found. The algorithm stops either when a maximum number of iterations is reached or when the number of reassignments falls below a predefined threshold (We chose early\_stop\_ratio = 0.0001).

The main data structures we utilized include:

* "assignments": A list where each index corresponds to a request and stores the "cid" - ID of its assigned cluster.
* "clusters": A mapping of cluster IDs to lists of request indices, representing the group of requests that belong to each cluster.
* "centroids": A structure holding the computed centroid vector for each cluster.

The clustering process produces a set of clusters, each containing semantically similar requests. Requests that do not meet the minimum cluster size requirement are categorized as unclustered.

**Cluster Naming**

We experimented with several approaches to assign meaningful names to the clusters.

Medoid-Based Naming:

* In our first attempt, we used a medoid-based approach, where we calculated the centroid of the embeddings within a cluster and selected the text of the request closest to this centroid as the cluster name.
* This method gave each cluster a long sentence as a name, and sometimes the sentences had more information than needed.

Frequency Analysis of Significant Words:

* We then tried a frequency-based approach, where we combined all the texts within a cluster, removed stopwords, and selected the most common words as the cluster name.
* While this method captured common terms effectively, it occasionally produced generic names due to the presence of frequent yet uninformative words.
* Additionally, it occasionally produced sentences that appeared strange and lacked proper syntax.

Final Method: KeyBERT with TF-IDF Backup

* After evaluating the outcomes of the previous methods, we selected a combined approach utilizing KeyBERT and TF-IDF.
* KeyBERT: Extracts key phrases based on BERT embeddings, focusing on semantically meaningful terms.
* TF-IDF Backup: In cases where KeyBERT failed to generate a suitable name, we applied TF-IDF analysis to identify the most significant words within the cluster.
* The Algorithm Logic:

1. Combine the texts within a cluster into a single document.
2. Apply KeyBERT to extract key phrases with length constraints.
3. If no valid phrase is found, apply TF-IDF to find the most significant words.
4. Assign the chosen phrase as the cluster name.

**Top-K Representative Sentences**

In our project, we incorporated a method to select the top-K representative sentences for each cluster.

Algorithm Overview

The selection of top-K representative sentences follows these steps:

1. For each sentence within a cluster, we used its embeddings.
2. We calculate the centroid of the cluster embeddings.
3. For each sentence, we compute the distance to the centroid using the Euclidean distance. This distance indicates how closely the sentence represents the cluster's overall semantic content.
4. We sort the sentences by their distance to the centroid, selecting the K sentences closest to the centroid.
5. To ensure diversity and avoid redundancy, we apply a semantic similarity threshold (we chose 0.93) using cosine similarity. Sentences that are too similar to already-selected ones are excluded.
6. Handling Edge Cases:

* If a cluster contains fewer than K sentences, all available sentences are selected.
* If a cluster has no valid embeddings, it is flagged as unprocessable.

The main data structures we utilized include:

* "Indices": Stores the indices of sentences belonging to the current cluster.
* "sentence\_emb\_pairs": Holds tuples where each element consists of a sentence (text) and its corresponding embedding.
* "cluster\_embeddings": A collection of all embeddings in the cluster, which is used to compute the centroid, representing the cluster’s central semantic meaning.
* "centroid": The mean vector of all sentence embeddings within the cluster.
* "distances": Stores tuples where each element contains: the sentence text, its euclidean distance from the centroid and Its embedding vector.
* "selected\_sentences": Stores the top-K representative sentences chosen based on their proximity to the centroid and semantic diversity constraints.
* "selected\_embeddings": Contains the embeddings of the selected representative sentences.
* "seen\_sentences": Tracks sentences that have already been added to prevent duplicate selections.

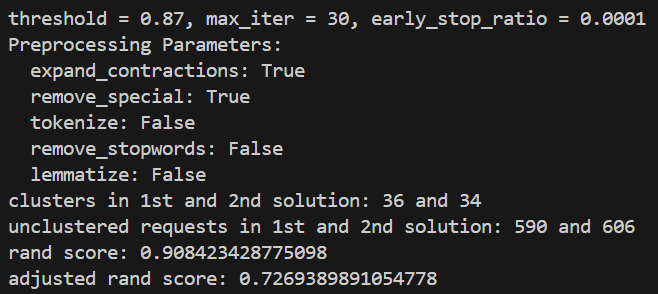
**Conclusion**

In conclusion, our project addressed the challenge of analyzing unrecognized user requests in goal-orienteddialog systems. By following the suggested algorithm from the project assignment, we developed a pipeline for text processing, clustering, cluster naming, and representative sentence selection.

We began with a flexible text-cleaning function that allowed parameterized configurations. The embeddings were generated using SentenceTransformer and normalized to improve clustering performance.

The clustering process utilized an iterative, distance-based approach without predefined clusters, dynamically forming groups based on Euclidean distance measurements. We explored multiple strategies for cluster naming, ultimately selecting a KeyBERT and TF-IDF hybrid approach to balance semantic relevance and linguistic clarity. And finally, we implemented the top-K representative sentence selection.

**Evaluation we got on the Covid19 dataset:**Execution Time: 42.79 seconds

****

**Evaluation we got on the Banking dataset:**Execution Time: 137.87 seconds

**תמונה שמכילה טקסט, צילום מסך, גופן

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.**