NLP's Final Project Report –

Analysis of Unrecognized User Requests in Goal-Oriented Dialog Systems

Data proccing:

Anticipated run time: ~ 1 min

The first thing we do with the input csv file is creating two objects:

1. *extracted_requests: a list of the requests (given index i, we can access the request by extracted_requests[i])*
2. *embedded_requests: a list of embedded requests using SBERT (the order of the list is the same as extracted_requests).*

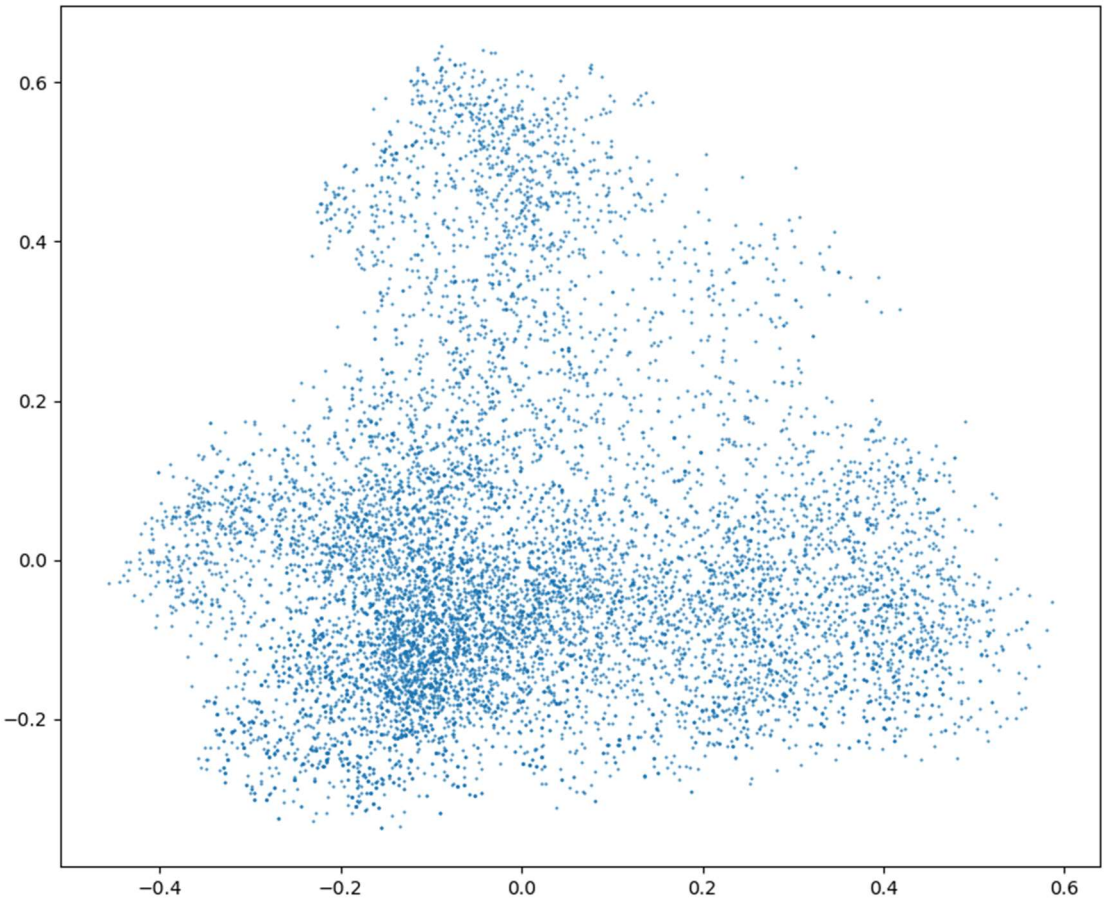
We chose `'all-mpnet-base-v2'` model to get the embeddings. (Runtime: ~2 min)

We had to choose values for two hyperparameters in this project:

1. The best suited pretrained SBERT model that was used to get request embeddings and the n-gram embeddings in part 3.
2. The best similarity_threshold (0.64) that determines "how close two vectors should be to be considered close enough to each other".

These hyperparameters were chosen after checking all possible combinations of the SBERT pretrained models with values for the threshold from the list [0.55, 0.56, 0.57, …, 0.9]. model_name = 'all-mpnet-base-v2', similarity_threshold = 0.64 yielded the highest ARI when tested on both datasets.

We also plotted the distribution of the embedded sentences using `PCA` algorithm to reduce the number of dimensions to 2.



Task 1: Clustering Requests

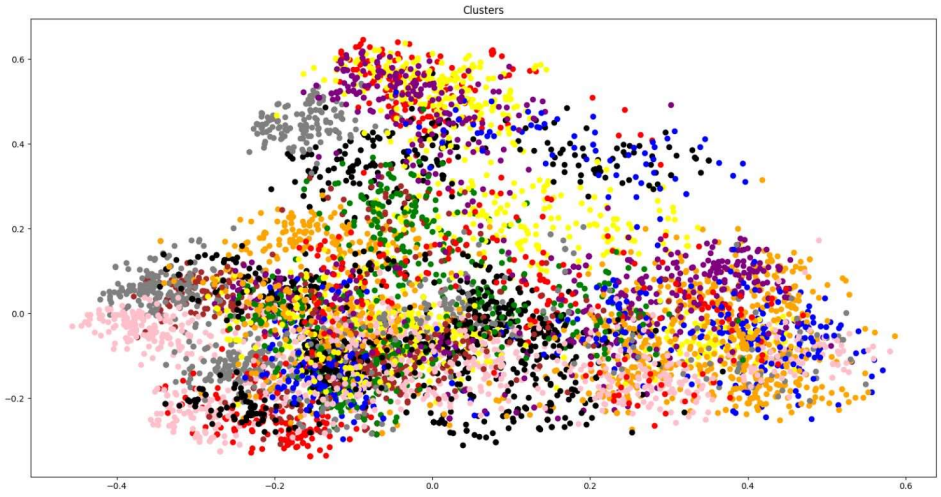
Anticipated run time: ~ 1.5 min

Our approach to the task:

As the project description suggested, the algorithm we implement is as follows:

Input: The embedded sentences, minimum similarity (threshold, determines how close the embeddings should be to be put in the same cluster) and minimum cluster size (discarding all clusters in which the number of elements is less than this number).
Output: Return the list of clusters, and each cluster has a list of sentences' embeddings and their request id's.

1. initialize clusters.
2. for each embedded sentence and its request id:
    - 2.1. for each cluster in clusters:
        - 2.1.1. calculate the similarity between the sentence and the cluster's centroid.
        - 2.1.2. if it is bigger than the maximum similarity:
            - 2.1.2.1. set this sentence as the maximum similarity.
    - 2.2. if the maximum similarity we got > minimum similarity we received (meaning we are above the similarity threshold):
        - 2.2.1. add the sentence's request id to the maximum cluster we just found.
        - 2.2.2. recalculate the cluster's centroid.
    - 2.3. otherwise:
        - 2.3.1. add a new cluster with the centroid=sentence's embedding
3. return the list of clusters (where each cluster is a list of sentences' embeddings and their request id's That their cluster size > minimum cluster size we received.



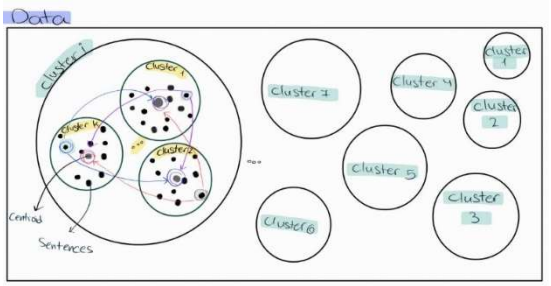
Task 2: Extraction Of Cluster Representatives

Anticipated run time: ~ 1 min

Our approach to the task:

We've tried various solutions, such as calculating the *cosine of the angle* between each combination of three sentences, calculating the distance of each sentence to every combination couple sentences. We also tried choosing 1 sentence randomly and then calculating the farthest sentence from it, and another sentence that Is the farthest away from them (by farthest we mean minimal *cosine angle*) which is a relatively easy solution, but we got worse results.

Eventually we chose the following algorithm:

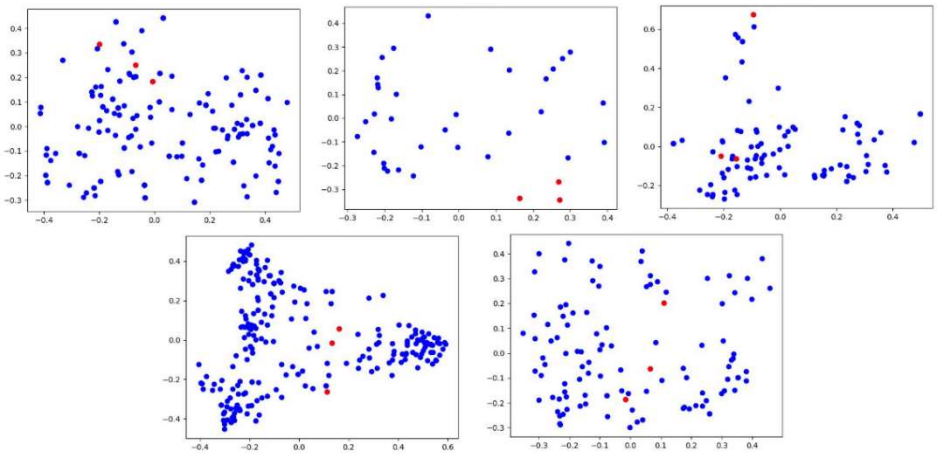


The idea: run KMeans on each "big" cluster (with k=number of representatives we want). Within each "kmeans" cluster, choose the embedding that's the furthest from all other "kmeans" clusters' centroids.

Input: The list of clusters (that will be referred from now on as *orig_clusters*) and number of representatives - k.

Output: Return the list of clusters, where each cluster has a list of k sentences id's that are the most diverse sentences in that cluster.

1. initialize empty list of clusters.
2. for each request:
    - 2.1. get the requsts' ids in that cluster.
    - 2.2. if the number of requests in the cluster is less than k:
        - 2.2.1. add the cluster to the list of clusters (we don't have other choice).
    - 2.3. else:
        - 2.3.1. run k-means on the cluster.
        - 2.3.2. get the centroids of the clusters.
        - 2.3.3. for each request in the cluster:
            - 2.3.3.1. compute the distance between the request and the centroid of the other clusters' centroids.
            - 2.3.3.2. add the request_id and the distance to a list.
        - 2.3.4. sort the list by the distance.
        - 2.3.5. add the first k requests to the list of clusters (these are the furthest from the other clusters' centroids).
3. return the list of clusters.



**Notice that the representatives don't look very far apart but we think (we hope) that it is because of the PCA algorithm, compresses it from multi-deletional to 2D.*

Task 3: Topic Modeling

Anticipated run time: ~ 12 min

Our approach to the task:

The first thing that we've tried was to use LDA models, but it didn't work very well. On top of that we tried using n-grams (and choose the one that is the closest to the cluster's centroid), pos-tagging (creating a well-formed sentence from the most frequent words in the cluster) and counting occurrences of all possible 2 to 5 n-grams and choosing the n-gram with the highest appearances in the corpus.

Eventually we chose the n-gram + embeddings-based algorithm below:

(*we chose n-grams of size 2-5 since we've tried several options like 6 and 7 n-grams but the result were worse)

Input: The list of clusters and their requests.

Output: Return the list of topics for each cluster.

1. initialize the cluster names list.
2. for each cluster:
    - 2.1. get the actual requests from the request_ids.
    - 2.2. create n-grams of the requests (2,3,4,5).
    - 2.3. remove empty lists.
    - 2.4. iterate over the list of lists:
        - 2.4.1. for each n-gram in the list of n-grams, join the words in the n-gram to form all the n-grams.
    - 2.5. embed the n-grams.
    - 2.6. check the cosine similarity between the n-grams and the centroid of the cluster.
    - 2.7. get the n-gram with the highest similarity.
    - 2.8. if the cluster name ends with a question mark, we remove the question mark and add "Questions about" at the beginning.
    - 2.9. add the cluster name to the list of cluster names.
3. return the cluster names list.

Evaluation:

Dataset 1:

RI Score: 0.867

ARI Score: 0.521

Dataset 2:

RI Score: 0.937

ARI Score: 0.487