Explore Exploit & Cold Start Report

In this assignment, I implemented multi-armed bandit, specifically epsilon-greedy, to address the cold start problem. The algorithm uses interaction calculated from *taxonomy* data for filtering out contents the selected user likes and calculating reward, and recommends content_id from *content* data. *taxonomy* has columns user_id, content_id, interaction, prompt, *content* has columns content_id, prompt, embeddings (bert embeddings of prompt and media_type), cluster (got from applying k-means clustering on embeddings).

calculate_interaction()

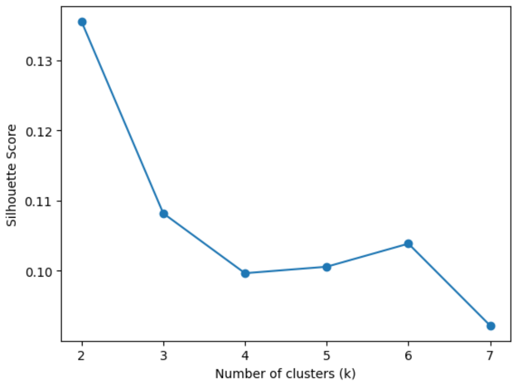
- The function calculate_interaction takes a single argument df, which is a DataFrame containing the columns: user_id, content_id, msEngagement, is_like, is_dislike
- The msEngagement values are scaled to a range between 0 and 1 using MinMaxScaler from sklearn. This normalization ensures that the engagement scores are on a consistent scale.
- The interaction score is calculated by combining is_like, is_dislike, and msEngagement:
    - is_like and is_dislike are subtracted to account for the difference between positive and negative feedback.
    - This difference is multiplied by 1.5 to weigh likes and dislikes more heavily than engagement.
    - The weighted difference is then added to msEngagement to get the overall interaction score.
- The interaction scores are again scaled to a range between -1 and 1 using MinMaxScaler.

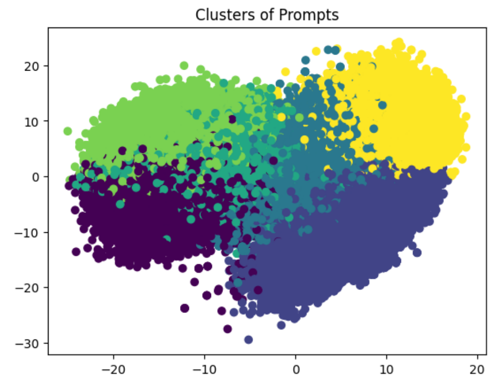
generate_embeddings()

- This function creates BERT embeddings for text prompts in a DataFrame and reduces their dimensionality.
- Each prompt in the dataframe is preprocessed using stemming, lemmatization and stopwords, removing unnecessary words and formatting.
- The get_embeddings function tokenizes and encodes each prompt using BERT, averaging the output embeddings to create a single vector for each prompt.
- The text embeddings are combined with dummy variables for media_type, the combined features are standardized. The embedding dimensionality is reduced using TruncatedSVD to 128 components and scaled with StandardScaler for consistency.
- The npy file of final embeddings is stored here.

Clustering

A silhouette analysis is used to evaluate the quality of K-Means clustering for different numbers of clusters (k) and the results are visualizes as below. k=2 yields the highest silhouette score, but small k values may not separate the contents clearly, so I chose k=6 in the end.





get_content()

- This is the candidate generator from the previous assignment.
- It recommends content for a specific user by finding and ranking similar items based on user interaction history and content similarity.
- It filters user_data to only include rows for the given user_id. If no data is found for the user, it returns None.
- It selects content that the user has interacted with positively (interaction > 0.49). 0.49 is chosen because I manually set interaction scores as 0.5 every time a new piece of content is recommended. If no such content is found, it returns None.
- Get user embeddings:
    1. It retrieves embeddings for content the user liked from *content*.
    2. It identifies the most-liked content and gets its cluster label.
- Find similar content:
    1. The function filters *content* to only include items within the same cluster as the most-liked content.
    2. It calculates cosine similarity scores between the embeddings of positively interacted content and all content in the cluster.
    3. It averages these similarity scores and sorts content by similarity in descending order.
- The function removes content that the user has already interacted with, and returns a list of content_id's, sliced to include the specified number_of_content items, starting from the given offset.

user_data

I also created a user_data global variable to store user "real-time" engagement data, which is passed as the first argument in get_content(). Since we don't really have real-time feedback from users, I just set interaction scores as 0.5 every time a new piece of content is recommended.

generate_popular_content()

- Selects a random piece of content from *taxonomy* with the highest interaction score.
- Updates user_data with an assumed interaction score of 0.5 to simulate real-time feedback.

generate_custom_content()

- Calls get_content() to generate personalized content based on the user's interaction history.
- Updates user_data with an assumed interaction score of 0.5 to simulate real-time feedback.

calculate_reward()

- Checks if the recommended content has an interaction record in taxonomy and returns the interaction score.
- If there's no interaction record, it uses synthetic data by averaging the user's past interactions within the same content cluster.

run_epsilon_greedy()

- This function implements epsilon-greedy algorithm.
- It initializes q_values and n_values for tracking the quality and counts of each bandit.
- With probability epsilon, the model explores, which means it chooses a random generator.
- With probability 1-epsilon, it exploits, which means it selects the generator with the highest estimated reward.
- It then calls either generate_popular_content() or generate_custom_content() based on the chosen bandit.
- It adjusts the estimated reward (q_values) for the chosen bandit.
- It records rewards and calculates average rewards over all trials.
- Expected value is calculated using:
    1. (1 - epsilon) * max(q_values): the reward from exploitation, where the algorithm selects the arm with the highest estimated reward (max(q_values)).
    2. epsilon * average_reward_all_arms: the reward from exploration, where the algorithm chooses a random arm with probability epsilon.
- It outputs rewards, average rewards, the expected value (EV), final Q-values, and a list of recommended content IDs.

The results are shown as below using user_id=00328ce57bbc14b33bd6695bc8eb32cdf2fb5f3a7d89ec14a42825e15d39df60, num_bandits = 2, num_trials = 50, and epsilon = 0.5.

```
Rewards: [-0.011642722250262598, -0.011642722250262598, -0.011642722250262598, -0.011642722250262598, -0.011642722250262598, -0.011642722
Average Rewards: [-0.011642722250262598, -0.011642722250262598, -0.011642722250262598, -0.011642722250262598, -0.011642722250262598, -0.0
Expected Value (EV): 0.008598080065423905
Estimated Q-values: [0.015345014170652739, -0.011642722250262598]
```

And the recommended content_id's are [129391, 125454, 126761, 128110, 125901, 127407, 126207, …].