Advanced Feature Engineering Guide

This guide outlines the strategy and implementation steps for creating four types of advanced features: co-click/sequential features, text embeddings, graph embeddings, and features from RNN models. The goal is to create powerful, non-customer-specific features from the historical supplementary datasets that can be merged into your main v9 training and test sets.

Technique 1: Co-Click & Sequential Features (Offer Relationships)

Goal: To understand which offers are frequently seen or clicked together. This captures the idea that "customers who like Offer A also tend to like Offer B."

Strategy: We will analyze the historical add_event log to find pairs of offers that appear in the same "session" (a short period of activity for a single customer).

- Step 1: Create Customer Sessions
  First, we need to group the raw event log into sessions. A session can be defined as all impressions for a single customer that occur within a specific time window (e.g., 30 minutes).
  ```
  # Pseudo-code for session creation
  import pandas as pd

  event_df = pd.read_parquet('/kaggle/input/amexkgp/add_event.parquet')
  event_df['impression_time'] = pd.to_datetime(event_df['id4'])
  event_df = event_df.sort_values(by=['id2', 'impression_time'])

  # A new session starts if the time since the last impression is > 30 minutes
  event_df['session_id'] = (event_df['impression_time'].diff().dt.total_seconds() > 1800).cumsum()

  print("Session IDs created.")
  ```
- Step 2: Calculate Co-occurrence & Co-click Matrices
  With sessions defined, we can now calculate which offers appear together. We can create a "co-click" matrix where the value for (Offer A, Offer B) is the number of times Offer B was clicked in a session where Offer A was seen.
- Step 3: Integrate as Features
  This co-click information can be aggregated for each offer to create powerful new features like:
  - avg_co_click_ctr: For a given offer, what is the average historical CTR of all

other offers that appeared in the same sessions?
    - top_co_click_offer_ctr: What is the historical CTR of the single most frequently co-clicked offer?

Technique 2: Text Embeddings (Offer Content Similarity)

Goal: To understand the semantic meaning of the offers themselves. This allows us to find offers that are similar in content, even if they have different IDs.

Strategy: We will use a powerful pre-trained language model to convert the text associated with each offer into a numerical vector (an embedding).

- Step 1: Prepare the Text Corpus
  Combine the text fields from offer_metadata.parquet (id9, f378) and the data_dictionary.csv (Description) to create a single descriptive text for each offer.
- Step 2: Generate Embeddings
  Use the sentence-transformers library, which provides easy access to state-of-the-art text embedding models.
  ```
  # Code snippet for generating embeddings
  from sentence_transformers import SentenceTransformer

  # Load a pre-trained model
  model = SentenceTransformer('all-MiniLM-L6-v2')

  # Assume 'offer_texts' is a list of the combined text for each offer
  offer_embeddings = model.encode(offer_texts)
  # This returns a numpy array where each row is a vector for an offer
  ```

- Step 3: Use Embeddings as Features
  The offer_embeddings array can be added directly to your dataset. Each dimension of the embedding vector becomes a new feature (e.g., text_emb_0, text_emb_1, ...). This gives your model a rich, nuanced understanding of the offer's content.

Technique 3: Graph Embeddings (Learning from the Network)

Goal: To capture an offer's "DNA" based on the network of customers who have interacted with it.

Strategy: We will create a bipartite graph of customers and offers and use a graph embedding algorithm like Node2Vec to learn a vector representation for each offer.

- Step 1: Construct the Bipartite Graph

Use the historical add_event log. Create a graph where every customer (id2) is one type of node and every offer (id3) is another. An edge exists between a customer and an offer if that customer has seen that offer.

- Step 2: Train a Graph Embedding Model
  Use a library like StellarGraph or node2vec to learn embeddings.
  ```
  # Pseudo-code for Node2Vec
  from node2vec import Node2Vec

  # G is your graph object from Step 1
  node2vec = Node2Vec(G, dimensions=64, walk_length=30, num_walks=200, workers=4)
  model = node2vec.fit(window=10, min_count=1, batch_words=4)

  # Get the vector for a specific offer
  # offer_vector = model.wv['offer_id_123']
  ```

- Step 3: Integrate Offer Embeddings
  Because of the "cold start" problem, we cannot use the learned customer embeddings. However, the offer embeddings are extremely valuable. They capture the essence of an offer based on the collective behavior of the historical customers who interacted with it. You can merge these offer embedding vectors into your v9 datasets using id3 as the key.

Technique 4: RNN for Sequential Patterns (Advanced)

Goal: To capture the sequential nature of how offers are viewed and clicked.

Strategy: While we can't directly model the sequences for our train/test customers, we can train an RNN on the historical add_event sessions to learn better offer embeddings.

- **The Model:** The model would take a sequence of offer IDs as input and try to predict the *next* offer ID that will be clicked in that session.
- **The Feature:** The real value is not the model's predictions, but the **learned weights in the offer embedding layer** of the RNN. These weights will capture complex sequential patterns (e.g., "Offer C is often clicked after Offer A and Offer B are seen but not clicked"). You can extract these learned embedding vectors and use them as features, just like with the text and graph embeddings.

By systematically implementing these advanced techniques, you will be creating a feature set that is far more powerful and nuanced than one based on simple

aggregates alone. This is the path to achieving a top score.