Fetch Rewards Take-home Report

Github: https://github.com/Qingyun-Wang/fetch_takehome

General approach

Our tool is designed to analyze customer input text and return related offers along with their similarity scores. Given the constraints of time and data availability, with only 384 offers in our dataset, it's impractical to develop an NLP model from scratch. Therefore, we have opted to leverage a pre-trained model to assess the similarity between the offers and the input texts.

To achieve this, we utilize the BertModel from the Transformers library, focusing on Word Embeddings. This approach allows us to effectively measure the degree of similarity between each offer and the customer's input. The process involves the following steps:

1) Clean up the offers and manage to assign the correct brand, retailer and category(and parent category if available) to offers. The assignment of category to offers is done by BertModel and cosine similarity.

2) Combine the brand, retailer and category together with the associated offer to create a new offer string. This will help us when the user input a specific brand, retailer or category.

3) Our method for determining similarity scores involves a dual-scoring system. The first component utilizes Fuzzywuzzy, specifically aimed at identifying key elements such as brand names or retailers within the text. The second component is based on the cosine similarity between two vectors: one representing the customer's input and the other representing our new offer text, as generated by the BertModel. This latter score captures the semantic similarity between the offers and the input text.To synthesize these two scores into a final similarity measure, we employ an algorithm that biases towards extreme values in either score. This method ensures that if either score is particularly high or low, it significantly influences the combined score. As a result, we can efficiently rank the offers in order of relevance to the customer's input

Data preprocessing

First, dealing with missing value. There are two part of missing value.

1) 146 out of 384 offer with a missing retailer information in the offer_retailer table. Either the offer don't have retailer associated with it or it's simply missing, I don't have easy way to retrieve the corresponding retailer so I will leave it for now.

2) 1 out of 9906 Brand is missing in the brand_category table, I simply drop it since just 1.

Secondly, combining tables.

We aim to categorize offers to provide users with relevant options in specific categories. This is achieved by performing a LEFT JOIN between the 'offer_retailer' and 'brand_category' tables, using the 'BRAND' as the key. If a brand corresponds to a single category, the categorization process is straightforward. However, approximately half of our brands are linked to multiple categories. In these cases, a simple join on the 'BRAND' column results in offers being associated with several categories. For such instances, we consider two potential solutions:

1) The offer is designed for use across multiple categories. This should work fine.

2) The offer is specific to a particular category within the range that the Brand encompasses. By leveraging the BertModel from the Transformers package, we can accurately assign the appropriate category to each offer. This is achieved by computing the similarity between the offer's description and the characteristics of each category.

Moreover, the LEFT joining between retailer table and brand_category create NULL values in the BRAND_BELONGS_TO_CATEGORY column since some of the brand don't have associated category. A couple techniques are examined to deal with this missing values.

1. Domain knowledge: since there are only a few BRAND involved. I can manual assign an appropriate Category from the list to it using my knowledge
2. Just use what's represents in BRAND to fill the NULL. with certain contextual information, BRAND sounds like an good proxy to category.

After we successful find the corresponding categories for each offer, now we can attach the brand, category, retailer informations into the offer string and get ready for comparison between offers and inputs.

Things to improve

There are several things we can do to improve the model. First, the search algorithm should be carefully selected and tested considering factors like keyword relevance, deal freshness, and user preferences. Also, it should allow for future improvements, such as implementing or upgrading indexing and ranking mechanisms to optimize search results. Second, user experience is also critical which can be enhanced by designing a user-friendly interface allowing users get clear and concise search results in a timely manner and implementing personalization features allowing users to set preferences, receive notifications for specific types of deals, or track favorite products. This enhances the user experience. As a product, the tool should encompass scalability approaches in the design, code, and infrastructure phases considering potential traffic growth. Since the tool provides offers, it should be ensured that users can always fetch real-time deals timely and accurately.

The next thing we can improve is a more sophisticated algorithm to better associate inputs with categories, and the relation between offers and categories(if direct data is not accessible). One of the way if we have more time and resources is using Transfer Learning: Transfer learning involves taking a model pre-trained on a large dataset and then fine-tuning it on a smaller, domain-specific dataset. This approach is particularly useful in our case that we don't have enough data for training. By attaching more layers into the pre-trained model, we are able to train our model and utilize the pre-trained model and make it works better specific to our task of relating inputs with categories.