**Use case – Personalized product search**

**Problem Scope:**

1. Personalization does not always improve product search quality. While some queries require high level of personalization, others might require a more general representation.
2. Current models unable to recommend good products –
   1. Unreliable personal information harm the search quality.
   2. Introduction of unnecessary noise.
   3. Do not conduct differential personalization adaptively under different context.

Hence, we require a much robust model to be able to scale the personalization based on the user query.

Features of the new model –

1. Dynamically control the influence of personalization
2. More flexible where personalization can vary from no to full effect.
3. considering the relation between historical purchased items.

Baseline model - ZAM model by Ai –

Limitation, ZAM model takes interaction between Query and item in history in a way that the max purchase history impact is = Mqu = Q + I while the least is Mqu = Q. But in real, Item can have more impact on the final recommendation and this is solved by TEM model where the weightage of Query and Item can be in any range

**Implementation details**

Data understanding and processing –

1. Used the data preparation steps as given in [Explainable-Product-Search-with-a-Dynamic-Relation-Embedding-Model/utils/AmazonDataset at master · QingyaoAi/Explainable-Product-Search-with-a-Dynamic-Relation-Embedding-Model](https://github.com/QingyaoAi/Explainable-Product-Search-with-a-Dynamic-Relation-Embedding-Model/tree/master/utils/AmazonDataset)
2. Used the data preparation steps to extract the relevant Amazon datasets into a trainable format.
3. Used Example.py to gather understanding of extracting amazon datasets.

Data Extraction and preparation for training: chosen single category -Cell Phones & Accessories

Create a bash script to extract and prepare dataset: ***old\ProdSearch\scripts\extract\_data.sh***

1. Download dataset:
2. Download meta data containing category info:
3. Remove stop words and preprocess data:
4. Indexing the dataset:
5. Matching the meta data with indexed data:
6. Splitting training and testing:

**Model development @ TEM**

1. Working components:
   1. Item Generation Model - Formally, let q be a query issued by a user u and i be an item in Si which is the set of all the items in a collection. Output - The probability of i being purchased by u given q.
   2. Query Representation - converting textual query into an embedding
   3. Item Representation- item embeddings are learned from their associated reviews.
2. Input to BERT:
   1. Query embedding
   2. Embedding of each Item (based on its review) arranges in new to old order (context of new to be stronger compared to older)
3. Output:
   1. q(l) is computed as a weighted combination of embeddings of query and purchased item

**Training the model:**

At training , We basically make the model learn based on

Configuration file - old\ProdSearch\scripts\config.py

Training file = python old\ProdSearch\scripts\train.py

**Evaluating the model:**

The end goal of this model is to produce a recommended item given a query. To do this, we extracted the ground truth purchase given a user query pair. We partition the purchases of a user to the training/validation/test set according to the ratio 0.8/0.1/0.1 in a chronological order. Our partition ensures that the purchases in the test set happen after the purchases in the training set.

We feed the Query embedding (Q0) and Historical items purchase (Iu) to the input. After multiple transformer layer, we use Q embedding output at layer ‘L’ (like <cls> token). This Q(l) is then flattened to produce weight associated with each item I belonging to a set of all Items in collection. Since Item embedding is actually the embedding created by the reviews given, we basically get weighted values of review embedding (corresponding to each item in collection)

**Metrics for evaluation –**

1. Mean Reciprocal Rank - MRR is a ranking metric used in information retrieval and recommender systems to evaluate how well a system ranks the first relevant result.
2. Precision

Table 1 – Comparison with Models @ 10 epochs: Unable to train different models due to GPU constraints

|  |  |  |
| --- | --- | --- |
| **Models** | **MRR** | **Precision** |
| ZAM |  |  |
| Review transformer |  |  |
| AEM |  |  |
| QEM |  |  |
| Item Transformer |  |  |

Table 2- Comparison with Epochs @ Item transformer

|  |  |  |
| --- | --- | --- |
| **Epoch** | **MRR** | **Precision** |
| 1 | 0.00024 | 0 |
| 5 | 4.94950e-05 | 0 |
| BEST model | 0.018 | 0.0028 |
|  |  |  |
|  |  |  |