LimeSpot Machine Learning Engineer Take-home assignment

Amir Hosein Safari

Octobor 15, 2021

1 Section 3

1.1

The recommender systems generally consist of two parts:

1. Candidate generation
2. Ranking

In Candidate generation, we filter our whole content catalog for a smaller portion of products that our users might be interested in during the candidate generation (or sourcing) phase. To do so, we must first map our items into a mathematical representation known as embeddings, which will allow us to utilize a similarity function to locate the items that are the most similar in space. This can be accomplished in a variety of ways.

1. Collaborative filtering: We put the information from items and users in a 2d Matrix and then we perform the matrix factorization to end up with user vectors and item vectors. However, for Collaborative filtering, the data sparsity is a key disadvantage.
2. word2vec for entities: The same as word embedding can be done for training embedding of user actions by treating sequence of user actions as context. For example, learning representations of items that were clicked or purchased.
3. graph learning: By viewing the recommendation process as a graph search, we use our model to generate recommendations by finding nodes that have high associations with the starting nodes. Many recommendation methods can be developed based on this graph view. Examples of these methods include direct retrieval, association mining, high degree association retrieval (You can see the related publication here).

In the ranking stage, we must design a ranking model that ranks items by their relevance after we have selected initial candidates for our suggestions. The purpose of this action, which can be expressed as a Machine Learning problem, is to optimize a tailored model for each user. This phase is critical because most interfaces have limited space for item recommendations, so we must make the most of it by placing the most relevant items at the very top.

My preferred model for this problem is the word2vec approach since it has lots of technical benefits such as scalability, relevancy, very low latency on real-time prediction, and catalog coverage. Since the recommender system must serve 100K interactions per minute, I think the word2vec approach is a very good choice.

There are two types of evaluations for recommender models:

1. Online evaluation which consists of an A/B testing approach. The metrics here are high-level matrices such as click-through rate and engagement.
2. Offline evaluation which is low-level metrics that are usually easily measurable. Such as precision and recall of the items we are suggesting to the user.

In this problem since we have a huge amount of interactions per minute in real-time, we need to consider these technical factors as well:

1. Scalability which means the algorithm is capable of training on a huge amount of data
2. Relevancy: The algorithm can be trained very quickly and frequently
3. Low latency: based on the user's most recent activities, it returns recommendations with very low latency.

1.2

1.2.1

In order to avoid overfitting the data, we can use the L2 regularization technique. We don't need L1 regularization, since in most cases the matrix is already sparse.

The way that L2 regularization works, is such that it adds a square magnitude of the coefficient to the loss function/error function.

1.2.2

Based on what I understand, we can do the clustering in matrix factorization if we make some changes to it and especially if we use the Non-negative matrix factorization. Therefore after creating the clusters, now if we want to find the outlier user, we calculate the distance between the user and its cluster. We have two options here:

1. If the distance is bigger than K, then we call that sample an outlier
2. If the distance is among the top K distance, then we call that sample outlier.

1.2.3

There are different ways to solve the cold start problem. But in general, we can divide the cold start problem into four categories:

1. Representative based
2. Content-based
3. Bandit
4. Deep learning

In the first approach, we can use the Representative Based Matrix Factorization (RBMF). RBMF is an extension of MF methods with an additional constraint that m items should be represented by a linear combination of k items. In this approach, when a new user joins the store, we can ask him/her to rate the k items and use that to infer the ratings of other $m - k$ items. This way, with a small additional cost on users rating some items, we can improve the recommendations for new users.

For instance, we know that Netflix is using the same approach for its recommendation service, and one way that they used to resolve the cold start problem is such that when a new user joins Netflix, it asked the new user about his/her preferences and interests.

Also, in my search, I found an interesting paper in which they tried to interview only a subset of users instead of all the new users to decrease the burden on the new users. You can find it here: https://dl.acm.org/doi/10.1145/3108148

In summary, the RBMF approach is the most suitable solution for this problem.

1.2.4

In order to add the shoppers and items attributes into the matrix factorization, we need to change some fundamental things.

We know the formula of matrix factorization without side information is as follow:

$X = A.B^T$ (I am ignoring other details of this formula)

In order to incorporate the attributes, we need to change the formula as follow:

$X = A.B^T$

$U = A.C^T$

$I = B.D^T$

U is a matrix representing side information about users, with each user being a row, and columns corresponding to their attributes.

I is similarly a matrix representing side information about items.

C and D are new latent factor matrices used for factorizing the side information matrices but are not used directly for X.

In this way, we can incorporate the user and item attributes in our prediction.

1.3

My understanding from this question is such that we have n store and each store has its own model. The items of each store are unique, but the shoppers can share with the stores.

Also, when we say that we want to recommend items from one store to another, it means that for instance if shopper A bought the sour beer from store X_1, we need to show shopper A, the similar products to sour beer in the store X_2 as well.

Giving the above definitions, I think we need to do the following:

1. We need to use the Item-to-Item Collaborative Filtering which means that except for finding similar shoppers, we find similar items and then make the recommendation based on the similar items.

2. We need to make a list of the cosine similarity of all items in all stores.

3. When shopper A makes interaction with item A_1 in store 1, we will find similar items to A_1 in all other stores, if the similarity is beyond some threshold (this is hyperparameter and need to be tuned) then we update the vectors in that store. For instance, assume A_1 similarity to A_2 in the store S_2 is more than the threshold. Then if a customer bought A_1 we update the vectors of S_2 in a way that shopper has interaction with the A_2.