Introduction to Recommender Systems course. Final project report

Alexander Kharitonov [*] **Konstantin Shlychkov** [*]
Danil Gusak [*] **Nikolay Kotoyants** [*] **Gleb Mazanov** [*]

Abstract

Martix factorization gained a lot of attention during last years. Most of all, for the ability to improve the speed and results of existing models. Modern improvements we can refer to such models as eALS (2), iALS (5) and others. Within this project we'll try to reproduce the work of the recommendation systems. In particular we're going to compare ALS, eALS, iALS and Graph Collaborative Filtering. We evaluate algorithms on ... data.

1. Introduction

There are many methods of solving the problem of building recommendation systems. Each model can differ both in the way the data are obtained (e.g., explicit and implicit feedback) and in the way the model is constructed. In this paper, we will try to list the main features of the models and demonstrate the results of their application to different datasets.

2. Problem statement

There is a matrix of user-item interactions $R \in \mathbb{R}^{M\times N}$, where M and N denote the number of user and items respectively. R denotes the set of user-item pairs whose values are non-zero. Vector p_u denotes the set of items that are interacted by u. Similar notations for q_i. Matrices $P \in \mathbb{R}^{M\times K}$ and $Q \in \mathbb{R}^{N\times K}$ the latent factor matrix for users and item.

Matrix factorization maps both users and items into a joint latent feature space of K dimension such that interactions are modeled as inner products in that space:

$$\hat{r}_{ui} = p_u^T q_i \tag{1}$$

The item recommendation problem is formulated as estimating the scoring function $\hat{r_u^i}$, which is used to rank items.

To learn model parameters in (3) introduced a weighted regression function, which associates a confidence to each prediction in the implicit feedback matrix R:

$$J = \sum_{u=1}^{m}\sum_{i=1}^{N} w_{ui}(r_{ui} - \hat{r}_{ui})^2 + \lambda(\sum_{u=1}^{m}||p_u||^2 + \sum_{i=1}^{N}||q_i||^2), \tag{2}$$

where w_{ui} weight of entry r_{ui}, λ controls the strength of regularization. Note that in implicit feedback learning, missing entries are usually assigned to a zero r_{ui} value but non-zero w_{ui} weight, both crucial to performance.

3. Related work

Methods of solving Item Recommendation Problem (GCP) include: Neighborhood models, Latent factor models, Alternating Least Square model (3), Element-Wise Alternating Least Squares model (2), Implicit Alternating Least Squares model (5), Neural Graph model (6). We are going to try to evaluate methods proposed in (3), (2), (5), (6) in different datasets.

4. Methods

4.1. Alternating Least Square model (block coordinate descent)

Let us introduce a set of binary variables p_{ui}, which indicates the preference of user u to item i. The p_{ui} values are derived by binarizing the r_{ui} values:

$$\text{p}_{ui} = \begin{cases} 1 & r_{ui} > 0 \\ 0 & r_{ui} = 0 \end{cases}$$

In other words, if a user u consumed item i (r_{ui}), then we have an indication that u likes i (p_{ui} = 1). On the other hand, if u never consumed i, we believe no preference (p_{ui} = 0). Also let's introduce a set of variables, c_{ui}, which measure our confidence in observing p_{ui}. A plausible choice for c_{ui} would be: $c_{ui} = 1 + \alpha r_{ui}$

Our goal is to find a vector $x_u \in R^f$ for each user u, and a vector $y_i \in R^f$ for each item i that will factor user preferences. In other words, preferences are assumed to be the inner products: $p_{ui} = x_u^T y_i$. These vectors will be known as the user-factors and the item-factors, respectively. Essentially, the vectors strive to map users and items into a common latent factor space where they can be directly compared. This is similar to matrix factorization techniques which are popular for explicit feedback data, with two important distinctions: (1) We need to account for the varying confidence levels, (2) Optimization should account for all possible u, i pairs, rather than only those corresponding to observed data. Accordingly, factors are computed by minimizing the following cost function:

$$min\sum_{u=1}^{m}\sum_{i=1}^{N}c_{ui}(p_{ui}-x_u^Ty_i)^2+\lambda(\sum_{u=1}^{m}||p_u||^2+\sum_{i=1}^{N}||q_i||^2), \tag{3}$$

The $\lambda(\sum\limits_{u=1}^{m}||p_u||^2+\sum\limits_{i=1}^{N}||q_i||^2)$ term is necessary for regularizing the model such that it will not overfit the training data. Exact value of the parameter λ is data-dependent and determined by cross validation.

The first step is recomputing all user factors. Let us assume that all item-factors are gathered within an $n \times f$ matrix Y. Before looping through all users, we compute the $f \times f$ matrix Y^TY in time $O(f^2n)$. For each user u, let usdefine the diagonal $n \times n$ matrix C_u where $C_{ii}^u = c_{ui}$, and also the vector $p(u) \in R^n$ n that contains all the preferences by u (the p_{ui} values). By differentiation we find an analytic expression for x_u that minimizes the cost function:

$$x_u = (Y^TC^uY+\lambda I)^{-1}Y^TC^up(u)$$

A recomputation of the user-factors is followed by a recomputation of all item-factors in a parallel fashion. We arrange all user-factors within an $m \times f$ matrix X. First we compute the $f \times f$ matrix X^TX in time $O(f^2m)$. For each item i, we define the diagonal $m \times m$ matrix C^i where $C_{ii}^u = c_{ui}$, and also the vector $p(i) \in R^m$ n that contains all the preferences by i. Then we solve:

$$y_i = (X^TC^iX+\lambda I)^{-1}X^TC^ip(i)$$

After computing the user- and item-factors, we recommend to user u the K available items with the largest value of $\hat{p}_{ui} = x_u^Ty_i$, where $\hat{p}_{ui}$ symbolizes the predicted preference of user u for item i. Full computational time complexity is $O(M+N)K^3+MNK^2$

4.2. element-wise Alternating Least Square model (eALS)

Matrix inversion is an expensive operation, usually assumed $O(K^3)$ in time complexity for ALS. To reduce the high time complexity, in (2) one applied a uniform weight to missing entries; i.e., assuming that all zero entries in R have a same weight w_0. Through this simplification, they can speed up the computation with memoization:

$$Q^TW^uQ = w_0Q^TQ+Q^T(W^u-W^0)Q, \tag{4}$$

where W^0 is a diagonal matrix that each diagonal element is w_0. As Q^TQ is independent of u, it can be pre-computed for updating all user latent vectors.

Also we can optimize parameters at the element level – optimizing each coordinate of the latent vector, while leaving the others fixed:

$$p_{uf} = \frac{\sum\limits_{i=1}^{N}(r_{ui}-\hat{r}_{ui}^f)w_{ui}q_{if}}{\sum\limits_{i=1}^{N}w_{ui}q_{if}^2+\lambda} \tag{5}$$

$$q_{if} = \frac{\sum\limits_{u=1}^{M}(r_{ui}-\hat{r}_{ui}^f)w_{ui}p_{uf}}{\sum\limits_{i=1}^{N}w_{ui}p_{uf}^2+\lambda} \tag{6}$$

All this transformations can reduce time complexity of computations to $O((M+N)K^2+|R|K)$

Algorithm 1 eALS

```
1: Input: $R, K, \lambda.W$ and item confidence vector $c$;
2: Output: Latent feature matrix $P$ and $Q$;
3: for $(u,i) \in R$ do
4:     $\hat{r}_{ui} \leftarrow p_u^Tq_i$
5: end for
6: while Stopping criteria is not met do
7:     $S^q = \sum_{i=1}^{N}c_iq_iq_i^T$
8:     for $u \leftarrow 1$ to $M$ do
9:         for $f \leftarrow 1$ to $K$ do
10:            for $i \in R_u$ do
11:                $\hat{r}_{ui}^f \leftarrow \hat{r}_{ui} - p_{uf}q_{if}$
12:            end for
13:            update $p_{uf}$
14:            for $i \in R_u$ do
15:                $\hat{r}_{ui} \leftarrow \hat{r}_{ui}^f + p_{uf}q_{if}$
16:            end for
17:        end for
18:    end for
19:    $S^p = P^TP$
20:    for $i \leftarrow 1$ to $N$ do
21:        for $f \leftarrow 1$ to $K$ do
22:            for $u \in R_i$ do
23:                $\hat{r}_{ui}^f \leftarrow \hat{r}_{ui} - p_{uf}q_{if}$
24:            end for
25:            update $q_{if}$
26:            for $i \in R_u$ do
27:                $\hat{r}_{ui} \leftarrow \hat{r}_{ui}^f + p_{uf}q_{if}$
28:            end for
29:        end for
30:    end for
31: end while
32: return $P$ and $Q$
```

4.3. Implicit Alternating Least Squares model (iALS)

The iALS algorithm targets the problem of learning an item recommender that is trained from implicit feedback. In this problem setting, items from a set I should be recommended to users $u \in U$. For learning such a recommender, a set of positive user-item pairs $S \subseteq U \times I$ is given. A recommender algorithm uses S to learn a scoring function $\hat{y} : U \times I \rightarrow R$ that assigns a score $\hat{y}(u,i)$ to each user-item pair (u, i). A common application of the scoring function is to return a ranked list of recommended items for a user u, e.g., sorting all items by $\hat{y}(u,i)$ and recommending the k highest ranked ones to the user.

The scoring function $\hat{y}$ of matrix factorization is defined as

$$\hat{y}(u,i) = w_u \times h_i$$

$W \in R^{U \times d}, H \in R^{I \times d}$ Here $d \in N$ is the embedding dimension, W is the user embedding matrix, H is the item embedding matrix

The iALS loss L(W, H) can be defined as:

$$L(W,H) = L_S(W,H) + L_I(W,H) + R(W,H) \tag{7}$$

$$L_S(W,H) = \sum_{u,i,y,\alpha \in S} \alpha(\hat{y}(u,i) - y)^2 \tag{8}$$

$$L_I(W,H) = \alpha_0 \sum_{u \in U, i \in I} (\hat{y})(u,i)^2 \tag{9}$$

$$R(W,H) = \lambda(||W||^2 + ||H||^2) \tag{10}$$

The first component L_S is defined over the observed pairs S and measures how much the predicted score differs from the observed label. component L_I is defined over all pairs in $U \times I$ and measures how much the predicted score differs from 0. The third component R is an L_2 regularizer that encourages small norms of the embedding vectors.

To optimize this objective we should optimize the user embeddings, W, and the item embeddings, H. When one side is fixed, e.g., H, the problem simplifies to $|U|$ independent linear regression problems, where w_u is optimized. The closed-form solution of each linear regression problem is:

$$w_{u^*} \leftarrow (\sum \alpha h_i \otimes h_i + \alpha_0 \sum h_i \otimes h_i + \lambda I)^{-1} \sum \alpha h_i y$$

A key observation for efficient computation is that the term

$$G^I = \sum h_i \otimes h_i$$

the Gramian of H, is shared between all users and can be precomputed. Algorithm 2 sketches the iALS algorithm. The computational complexity to optimize all user embeddings is $O(d^2|S| + d^3|U|)$. Learning the item side is analogous to the user side with the same complexity and the overall complexity becomes $O(d^2|S| + d^3(|U| + |I|))$.

Algorithm 1 iALS algorithm [7]

```
1: for $t \in \{1, \ldots, T\}$ do
2:     $G^I \leftarrow \sum_{i \in I} \mathbf{h}_i \otimes \mathbf{h}_i$                          ▷ $\mathcal{O}(|I|d^2)$
3:     for $u^* \in U$ do
4:         $\nabla_{\mathbf{w}_{u^*}} \leftarrow 0$
5:         $\nabla^2_{\mathbf{w}_{u^*}} \leftarrow \alpha_0 G + \lambda$                        ▷ $\mathcal{O}(d^2)$
6:         for $(u^*, i, y, \alpha) \in S$ do
7:             $\nabla_{\mathbf{w}_{u^*}} \leftarrow \nabla_{\mathbf{w}_{u^*}} + \alpha y \mathbf{h}_i$             ▷ $\mathcal{O}(d)$
8:             $\nabla^2_{\mathbf{w}_{u^*}} \leftarrow \nabla^2_{\mathbf{w}_{u^*}} + \alpha \mathbf{h}_i \otimes \mathbf{h}_i$   ▷ $\mathcal{O}(d^2)$
9:         end for
10:        $\mathbf{w}_{u^*,\boldsymbol{\pi}} \leftarrow (\nabla^2_{\mathbf{w}_{u^*}})^{-1} \nabla_{\mathbf{w}_{u^*}}$            ▷ $\mathcal{O}(d^3)$
11:    end for
12:    Perform a similar pass over the item side
13: end for
```

Hyperparameters

It is advisable to measure the metrics during training after each iteration. This removes **the number of iterations** T from the search space – provided that T is large enough. A too large value of T is not a concern with respect to quality, but only with respect to runtime. iALS onverges usually within a few iterations and a reasonable initial choice could be 16 iterations.

Both **unobserved weight** α_0 and the **regularization** λ are crucial for iALS and it is important to choose them carefully. It is advisable to search the unobserved weight together with the regularization because both of them control the trade-off between the three loss components, L_S, L_I and R. Intuitively, we know that for item recommendation we need both L_S and L_I. So, the observed error values of L_S and L_I should not differ by several orders of magnitude. Similarly, with large embedding dimensions, we need some regularization, so again the values of R, L_S and L_I should have comparable orders of magnitude.

4.4. Neural Graph Collaborative Filtering (NGCF)

Neural Graph Colloborative Filtering (NGCF) is a recommendation system that combines neural networks and graph algorithms to model user-item interactions in recommender systems. The NGCF model represents the user-item interactions as a bipartite graph, where users and items are nodes in the graph, and edges represent the interactions between users and items.

The NGCF model employs graph convolutional neural networks (GCNs) to learn the latent representations of users and items. GCNs are a type of neural network that operates directly on graph structures, allowing them to incorporate the local connectivity patterns of the graph into the model. By using GCNs, NGCF can capture the complex relationships between users and items, and make more accurate recommendations.

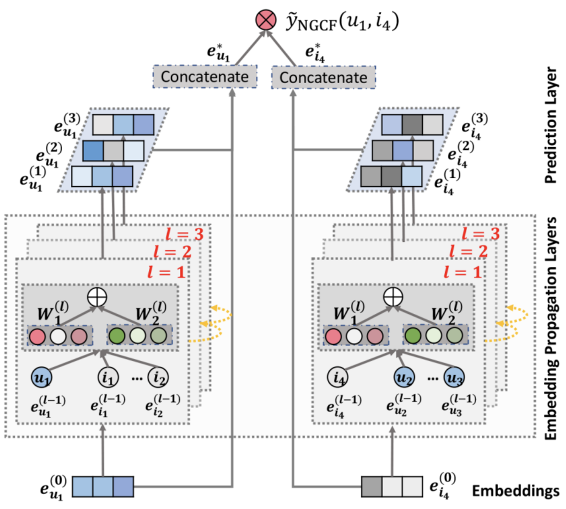

Figure 1. An illustration of NGCF model architecture (the arrowed lines present the flow of information). The representations of useru1 (left) and itemi4 (right) are refined with multiple embedding propagation layers, whose outputs are concatenated to make the final prediction

NGCF also uses a hybrid approach to combine collaborative filtering and content-based methods. In addition to user-item interactions, the model can also incorporate features of users and items, such as user demographics, item attributes, and textual descriptions. These features are used to enhance the recommendation performance, especially when the user-item interaction data is sparse or noisy.

Following mainstream recommender models , we describe a user u (an item i) with an embedding vector $e_u \in R^d (e_i \in R^d)$, where d denotes the embedding size. This can be seen as building a parameter matrix as an embedding look-up table:

$$E = [\underbrace{e_{u_1}, ..., e_{u_N}}_{user\ embeddings}, \underbrace{e_{i_1}, ..., e_{i_N}}_{item\ embeddings}] \tag{11}$$

Message Construction. For a connected user-item pair (u, i), we define the message from i to u as:

$$m_{u \leftarrow i} = \frac{1}{\sqrt{|N_u||N_i|}} \Big(W_1 e_i + W_2 (e_i \otimes e_u) \Big) \tag{12}$$

,where $m_{u \leftarrow i}$ is the message embedding (i.e., the information to be propagated). Encoding function, takes embeddings e_i and e_u as input, and uses the coefficient p_{ui} to control the decay factor on each propagation on edge (u, i). $W1, W2 \in R^{dd}$ are the trainable weight matrices to distill useful information for propagation, and d is the transformation size. $e_i \otimes e_u$ denotes the element-wise product

Message Aggregation. In this stage, we aggregate the messages propagated from u's neighborhood to refine u's representation. Specifically, we define the aggregation function as:

$$e_u = LeackyReLu\Big(m_{u \leftarrow i} + \sum_{i \in N_u} m_{u \leftarrow i}\Big) \tag{13}$$

With the representations augmented by first-order connectivity modeling, we can stack more embedding propagation layers to explore the high-order connectivity information. Such high-order connectivities are crucial to encode the collaborative signal to estimate the relevance score between a user and item.

By stacking l embedding propagation layers, a user (and an item) is capable of receiving the messages propagated from its l-hop neighbors. In the l-th step, the representation of user u is recursively formulated as:

$$e_u^l = LeackyReLu\Big(m_{u \leftarrow i}^l + \sum_{i \in N_u} m_{u \leftarrow i}^l\Big) \tag{14}$$

wherein the messages being propagated are defined as follows,

$$\begin{cases} m_{u \leftarrow i}^l = p_{ui}\big(W_1^l e_i^{l-1} + W_2^l (e_i^{l-1} \otimes e_u^{l-1})\big) \\ m_{u \leftarrow i}^l = W_1^l e_i^{l-1} \end{cases} \tag{15}$$

, where $W_1^l, W_2^l, \in R^{d_l d_{l-1}}$ are the trainable transformation matrices, and d_l is the transformation size; e_i^{l-1} is the item representation generated from the previous message-passing steps, memorizing the messages from its $(l-1)$-hop neighbors. It further contributes to the representation of user u at layer l. Analogously, we can obtain the representation for item i at the layer l.

To learn model parameters, we optimize the pairwise BPR loss, which has been intensively used in recommender systems . It considers the relative order between observed and unobserved user-item interactions. Specifically, BPR assumes that the observed interactions, which are more reflective of a user's preferences, should be assigned higher prediction values than unobserved ones. The objective function is as follows

$$Loss = \sum (-ln(\sigma(\hat{y_{ui}} - \hat{y_{uj}})) + \lambda ||\Theta||_2^2 \tag{16}$$

5. Experiments

All code can be found here: Github

5.1. Datasets and Evaluation Protocol

We evaluate on two publicly accessible datasets: Yelp and Movielens 1M.

We split each dataset into training, testset, and holdout datasets based on a timepoint split with quantile value and according to the ”warm-start” evaluation strategy: the hold-out dataset contains only the immediate interactions following the fixed timepoint for each test user from the testset; the set of users in training part is disjoint with the set of users in the testset, which implements the ”warm-start” scenario. The evaluation metrics are HR@10 (hit rate), MRR@10 (mean reciprocal rank), nDGC@10 (normalized discounted cumulative gain) and COV@10 (coverage).

5.2. Results

5.2.1. Matrix Factorizations

To conduct experiments with ALS, eALS and iALS algorithms we decided to implement all of them, using Pytorch(4) framework. We have managed to fully vectorize ALS and use all power of modern computetional devices. Our eALS and iALS implementations have `for` loop in pure Python and therefore, significantly increasing calculation time. To train our algorithms we set the size of user’s and item’s embeddings $K = 100$ and regularization factor $\lambda = 0.1$. For eALS and iALS algorithms we used equal weights for all unseen interactions.

To make our study more holistic, we ran the training and evaluation processes for ALS and BPR recommendation models from Implicit(1) package. The resulting values of metrics can be found in Figures 2, 3 for Movielens 1M and Yelp datasets respectively. The resulting time consumption of algorithms can be found on figures 4, 5, 6. Few interesting conclusions follows from these plots:

1. NDCG outperforms only BPR model from Implicit package according to HR, MRR and nDCG metrics, that is the same results as in the article. However NDCG shows much worse results in comparison with Matrix Factorizations methods. Meanwhile, NGCG totally beat all other models in terms of COV. It might mean, that graph-based approach managed to find non-trivial dependencies between interests of different users and make recommendation more distinguish without enough respect for user’s preferences.

2. ALS from Implicit package shows the best performance on Movielens dataset in terms of HR, MRR, nDCG. At the same time, our own implementation of ALS outperform other algorithms on the YELP datasets in respect to the HR, MRR, nDCG. It was surprising to us, that ALS approach, which we have considered as a baseline shows the best performance.

3. Due to vectorization of ALS, computetional time of this algorithm is very fast in comparison to eALS and iALS, which should be faster in theory, but slower in practice.

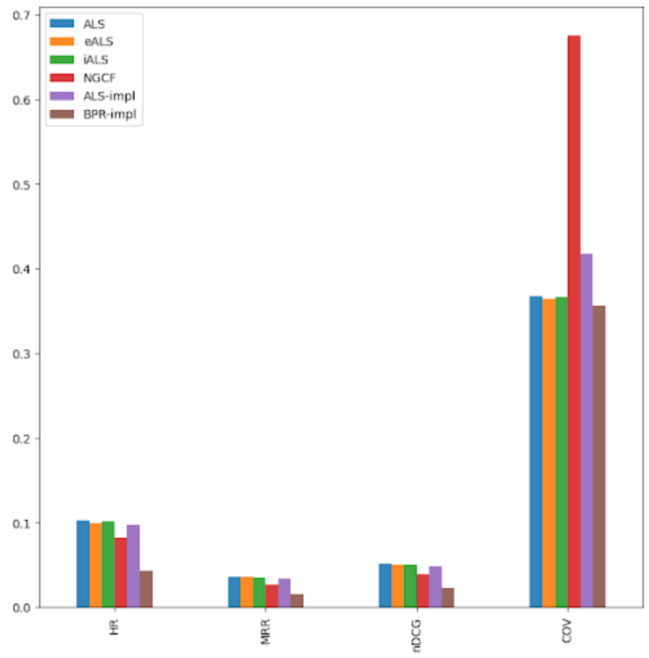

Figure 2. Values of the metrics obtained on the MovieLens dataset

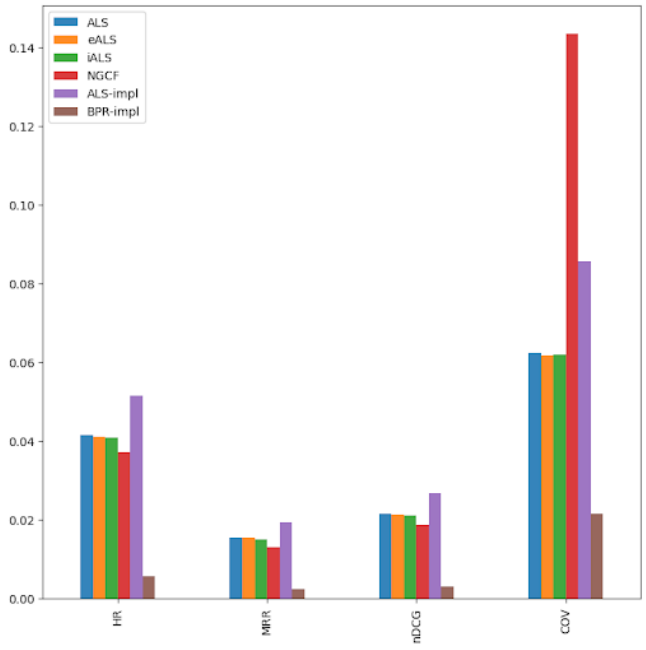

Figure 3. Values of the metrics obtained on the Yelp dataset

5.2.2. Graph Neural Network

For all experiments we followed the same setup as in the original paper(6) but didn’t apply a grid search for hyperparameters and choose best hyperparameters obtained by authors, namely the embedding size was fixed to 64 for all layers, learning rate was set 0.005, regularization parameter

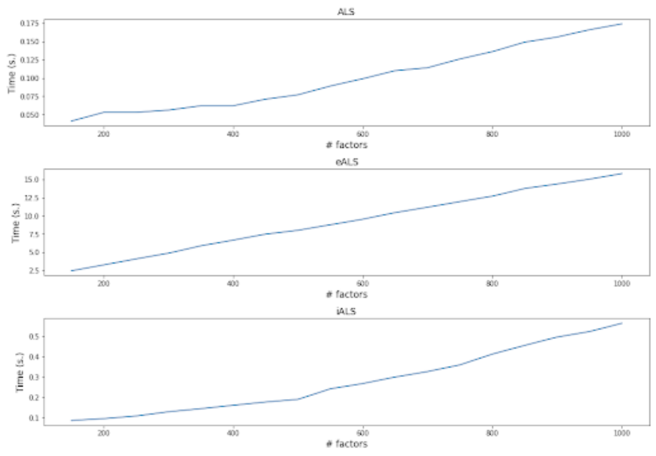

Figure 4. Time complexities graphs for ALS, eALS, iALS models

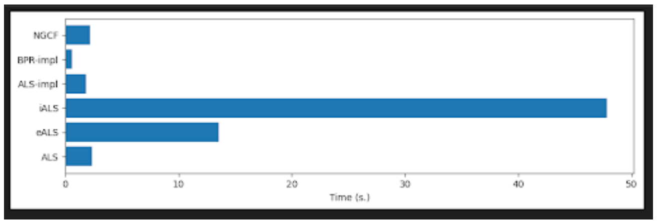

Figure 5. Computational speed graphs for MovieLens dataset

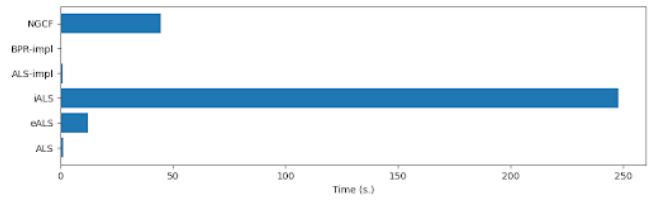

Figure 6. Computational speed graphs for Yelp dataset

for BPR loss was chosen 10^{-5}. For the MovieLens dataset we used the same batch of size 1024 and trained model for 400 epochs. For the Yelp dataset, due to computational resources limit, we set batch size 512 and trained model for only 30 epochs.

We successfully reproduced the results from (6) — graph neural network showed the better performance compared to BPR model, but we obtained worse performance with NGCF model than with other matrix factorization models. The gap in metrics between NGCF and ALS — best matrix factorization model in our experiments — though, is not so wide as between ALS and BPR and we actually boost NGCF quality in our experiments. This difference in performance may be explained the poor choice of loss function — BPR loss for both NGCF and BPR model and the possible improvement for the neural network model may be the usage of another loss function, for instance, the we can use ALS or iALS loss functions or WARP loss.

6. Conclusions and future work

In our work, we have accomplished all the tasks we set out to do. We revised the article and reproduced its contents. We have implemented a few different Collaborative Filtering models, namely ALS, eALS, iALS, NGCF. We have compared computational time and the results on different metrics. As further work, it is necessary to implement the iALS++ model, another neural model, and compare the results on more datasets to get a complete picture of the results of the methods presented above. In our future we plan to train the graph neural network with another loss function, for instance, ALS loss function or WARP loss. Also we want to investigate the convergence speed for all presented algorithms and fine-tune our models applying grid-search.

References

[1] Implicit: Fast python collaborative filtering for implicit datasets.

[2] Xiangnan He, Hanwang Zhang, Min-Yen Kan, and Tat-Seng Chua. Fast matrix factorization for online recommendation with implicit feedback. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 549–558, 2016.

[3] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE international conference on data mining*, pages 263–272. Ieee, 2008.

[4] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[5] Steffen Rendle, Walid Krichene, Li Zhang, and Yehuda Koren. Revisiting the performance of ials on item recommendation benchmarks. In *Proceedings of the 16th ACM Conference on Recommender Systems*, pages 427–435, 2022.

[6] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR con-*

ference on Research and development in Information Retrieval, pages 165–174, 2019.