Higher Dimensional Collaborative Filtering for Beer Networks

Chris Jerrett, Athreya Murali, Ujwal Pandey
Computer Science Department
Rensselaer Polytechnic Institute
Troy, NY, USA

Abstract

We introduce a new method for collaborative filtering for recommender systems with vector weight edges. In this paper we detail how to compute tensor decomposition for a special adjacency tensors and detail the results.

I. Introduction

Recommender systems have been widely used for creating personalized predictive models that help individuals identify content of interest. Collaborative filtering (CF) is the '*de-facto*' standard used for these systems. Matrix Factorization (MF) is a popular method to perform CF and has been used in many popular systems i.e. Netflix Movie Prize [1]. The general problem set-up considers a set of users $U = \{u_1, u_2, .., u_n\}$, a set of reviews of a product $\{r_1, r_2, ..., r_m\}$ and set of ratings for the product $\{w_1, w_2, .., w_m\}$ where $w_i \in \mathbb{R}$. This set-up can be viewed as a bipartite graph $G_{X,Y} = (V, E)$ and the goal is to predict edges and their weights. Using MF, we consider the adjacency matrix $A \in \mathbb{R}^{n \times m}$ to solve the following problem

$$\min_{U,V} \left\| A - UV^T \right\|_F^2$$

where $U \in R^{n \times k}$ is the user-feature matrix and $V \in R^{m \times k}$ is the product-feature matrix. There are many methods to solve this problem such as Singular-Value Decomposition (SVD), Stochastic Gradient Descent (SGD), and Weighted Alternating Least-Squares (WALS). However, issues of sparsity of the dataset tend to make this problem a little more difficult to solve.

In this paper, we consider a variant of this problem; we consider a product with a multi-attribute rating. The problem changes slightly since our set of ratings $\{w_1, w_2, .., w_m\}$ now has $w_i \in \mathbb{R}^k$. To model this problem, we now consider the adjacency tensor $\mathcal{A} \in \mathbb{R}^{n \times m \times k}$, a real 3-dimensional array, where its element is denoted by $\mathcal{A}_{u_i, r_i} = w_i$. In this paper, we explore mathematically and experiment with Masked CP-Decomposition, a tensor decomposition method, on the BeerAdvocate dataset provided by SNAP Labs. We discuss the pre-processing of the dataset, capture the experiments by measuring the error of method using both proximal gradient descent and stochastic gradient descent, and compare our results to a simplistic model.

II. Background

We start by presenting a few definitions required to understand this work [2].

Definition II.1 (Simple Tensor)**.** A simple tensor or order n is a tensor $\mathcal{X}$ such that there exists n vectors $X_1, X_2, ..., X_n$ such that $\mathcal{X} = X_1 \otimes X_2 \otimes ... \otimes X_n$. Where $\otimes$ is the outer product.

Definition II.2 (Tensor Rank)**.** The rank of a tensor $\mathcal{X}$ is the minimum number of simple tensors $\mathcal{X}_1, \mathcal{X}_2, ... \mathcal{X}_F$ such that $\mathcal{X} = \mathcal{X}_1 + \mathcal{X}_2 + ... + \mathcal{X}_F$.

Definition II.3 (CP Decomposition)**.** The CANDECOMP/PARAFAC (CP) decomposition of an order N tensor $\mathcal{X}$ is a tensor rank F approximation of $\mathcal{X}$ such that,

$$\begin{aligned}\mathcal{X} &\approx [\![U_1, U_2, ..., U_N]\!] \\ &= \sum_{i=1}^{F} U_1(:, i) \otimes U_2(:, i) \otimes ... \otimes U_N(:, i)\end{aligned}$$

where $U_i(:, j)$ is the jth column of the ith feature matrix.

Definition II.4 (Mode-n unfolding)**.** The mode-n unfolding of a tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times ... I_N}$ is a matrix $X \in \mathbb{R}^{I_n \times I_1 I_2 ... I_{n-1} I_{n+1} ... I_N}$ such that

$$\mathcal{X}(i_1, i_2, ..., i_N) = \mathcal{X}_{(n)}(j, i_n)$$

with $j = 1 + \sum_{k=1, k \neq n}^{N} (i_k - 1) j_k$ and $j = \prod_{m=1, m \neq k}^{k-1} I_m$.

We note that the mode n unfolding of a tensor is a way to effectively turn the tensor into a matrix by concatenating mode-n slices.

Definition II.5 (Mode-n fiber)**.** The mode-n fiber is the vector obtained by fixing all indices except for the i-th index.

$$\mathcal{X}(i_1, i_2, ... i_{n-1}, :, i_{n+1}, ..., i_N)$$

We can describe the dataset using a weighted bipartite graph G. We have an edge e between a product $b \in B$ and a user $u \in U$ with $e = (u, b)$ and $w(e) \in \mathbb{R}^k_{\geq 0}$ where k is the number of attributes users rate each product on. We will represent the data using a adjacency tensor $\mathcal{A} \in \mathbb{R}^{|B|+|U| \times |B|+|U| \times k}$. If we look at the structure a frontal slice of $\mathcal{A}$ we see that $\mathcal{A}$ has the following block diagonal structure

$$\begin{bmatrix} A & 0 \\ 0 & A^\top \end{bmatrix}.$$

Thus we only have to store $A \in \mathbb{R}^{|B| \times |U| \times k}$ and achieve a speed up by a factor of 4. Thus, we can define the adjacency tensor of our dataset as

$$\mathcal{A}(u, b, k) = \begin{cases} w((u, b))_k & \text{if } (u, b) \in G \\ 0 & \text{if } (u, b) \notin G \end{cases} \tag{1}$$

Alternating Least Squares (ALS), the traditional method of computing the rank R CP-Decomposition of a order m tensor $\mathcal{X}$, relies on updating each factor of the matrices $A_{(n)}$ by solving the following optimization problem [4].

$$A_{(n)}^{(r)} = \underset{A}{\operatorname{argmin}} \left\| \mathcal{X}_{(n)} - H_{(n)} A^\top \right\|_F^2 \tag{2}$$

where $\mathcal{X}_{(n)}$ is the mode N unfolding of the tensor $\mathcal{X}$ and $H_{(n)} = A_{(1)} \circ A_{(2)} \circ ... \circ A_{(n-1)} \circ A_{(n)} \circ ... \circ A_{(N)}$. We note that computing $H_{(n)} A_{(x)}$ is computing our rank F estimate of $\mathcal{X}_{(n)}$. The ALS optimization problem given above has a closed form solution given by

$$\left(\left(H_{(n)}^\top H_{(n)} \right)^{-1} H_{(n)}^\top \mathcal{X}_{(n)} \right). \tag{3}$$

Computing $\left(H_{(n)}^\top H_{(n)} \right)^{-1}$ is relatively simple but as $\mathcal{X}$ gets larger computing the decomposition becomes infeasible. If $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times ... \times I_m}$ then computing the matricized tensor times Khatri-Rao product (MTTKRP) $H_{(n)}^\top \mathcal{X}_{(n)}$ requires $O\left(\prod_{n=1}^{N} I_n F \right)$ operations.

III. Algorithms

A. Stochastic Gradient Descent

In [3] an algorithm for computing the decomposition by randomly sampling fibers of the tensor $\mathcal{X}$ is given by the following algorithm.

First, we sample a set of mode n fibers of $\mathcal{X}$ where each fibers is a row of the mode-n unfolding of the tensor $\mathcal{X}$. We denote the set of mode n fibers sampled from $\mathcal{X}$ as $\mathcal{F}_n \subset \{1, 2, 3, ...J_n\}$. If $|\mathcal{F}_n| > F$ then we can solve the sketched system of equations,

$$A_{(n)}^{(i)} = \underset{A}{\operatorname{argmin}} \left\| \mathcal{X}_{(n)}(\mathcal{F}_n, :) - H_{(n)}(\mathcal{F}_n, :)A^{\intercal} \right\|_F^2 \tag{4}$$

by computing,

$$A_{(n)}^{\intercal(i)} = H_{(n)}(\mathcal{F}_n, :)^{\dagger} \mathcal{X}_{(n)}(\mathcal{F}_n, :). \tag{5}$$

We could utilize this method but it does not allow for constraining the factors $A_{(n)}$. Instead, we consider the stochastic gradient for each factor matrix given by,

$$G_{(n)}^{(r)} = \frac{1}{|\mathcal{F}_n|} \left(A_{(n)}^{(r)} H_{(n)}^T(\mathcal{F}_n) H_{(n)}(\mathcal{F}_n) - \mathcal{X}_{(n)}(\mathcal{F}_n) H_{(n)}(\mathcal{F}_n) \right) \tag{6}$$

$$G_{(n')}^{(r)} = 0, n' \neq n. \tag{7}$$

We can then adaptively select the step size using adagrad [7].

B. Constrained Case

If we want to constrain each factor matrix $A_{(n)}$ to some convex set C we can apply proximal gradient descent to the convex $\infty - 0$ indicator function ι_C defined as

$$\iota_C(x) = \begin{cases} 0 & \text{if } x \in C \\ \infty & \text{if } x \notin C \end{cases}. \tag{8}$$

Then on each iteration project A_n into C by computing

$$\operatorname{prox}(A_n) = \underset{A}{\operatorname{argmin}} \left(\iota_C(a) + \frac{1}{2} \left\| A - A_{(n)} \right\|_2^2 \right) \tag{9}$$

$$= \underset{A \in C}{\operatorname{argmin}} \left\| A - A_{(n)} \right\|_F^2. \tag{10}$$

We then set update $A_n^{(r+1)}$ as

$$A_{(n)}^{(r+1)} \leftarrow \operatorname{prox}\left(A_{(n)}^{(r)} - \alpha G_{(n)}^{(r)} \right) \tag{11}$$

where α is the step size selected by adagrad.

C. Masked-CP Decomposition

If we try to solve

$$\min \left\| \mathcal{X}_{(n)} - H_{(n)} A_{(n)}^{\intercal} \right\|_F^2 \tag{12}$$

then the problem is dominated by entries with value 0 in $\mathcal{X}$ where the review is missing. If we naively solve this problem, we will not get good generalization. Thus we introduce the concept of a mask tensor

$\mathcal{M}$ which has the same dimensions as $\mathcal{X}$. $M[u,v,:] = \mathbf{0}$ if we are missing the review user u gave product v and $\mathbf{1}$ otherwise. We then try to solve the related problem

$$\min \left\| M \circ \left(\mathcal{X}_{(n)} - H_{(n)} A_{(n)}^{\mathsf{T}} \right) \right\|_F^2 \tag{13}$$

$$= \min \left\| \mathcal{X}_{(n)} - M \circ \left(H_{(n)} A_{(n)}^{\mathsf{T}} \right) \right\|_F^2 . \tag{14}$$

We can then obtain an analogous gradient as before

$$G_{(n)}^{(r)} = \frac{1}{|\mathcal{F}_n|} \left(\left(M^T(\mathcal{F}_n) \circ \left(A_{(n)}^{(r)} H_{(n)}^T(\mathcal{F}_n) \right) \right) H_{(n)}(\mathcal{F}_n) - \mathcal{X}_{(n)}(\mathcal{F}_n) H_{(n)}(\mathcal{F}_n) \right) \tag{15}$$

apply the same two algorithm as before. The new optimization problem will no longer penalize entries that we do not have data for so the value of $\mathcal{X}(u,v,:)$ will no longer influence $A_{(n)}$ if (u,v) is not in the dataset.

IV. Methodology

A. Procuring Data

The dataset used for this project is a collection of about 1.5 million reviews of beers over a period of 10 years from the website BeerAdvocate. The dataset is provided by the Stanford SNAP Lab and is hosted on data.world.

The dataset is stored as a CSV file. Each row of the file contains the following columns

1) *brewery_id*: unique numerical identifier for brewery that manufactured reviewed beer
2) *brewery_name*: name of brewery that manufactured reviewed beer
3) *review_time*: time of review (as Unix time)
4) *review_overall*: overall rating of beer by reviewer, on scale of 1 to 5 increments of 0.5
5) *review_aroma*: rating of beer aroma by reviewer, on scale of 1 to 5 increments of 0.5
6) *review_appearance*: rating of beer appearance by reviewer, on scale of 1 to 5 increments of 0.5
7) *review_profilename*: Profile name of reviewer
8) *beer_style*: category of beer (one of 104 possible categories, e.g.,"Light Lager")
9) *review_palate*: rating of beer palate by reviewer, on scale of 1 to 5 increments of 0.5
10) *review_taste*: rating of beer taste by reviewer, on scale of 1 to 5 increments of 0.5
11) *beer_name*: name of reviewed beer
12) *beer_abv*: percent alcohol by volume of beer
13) *beer_beerid*: unique numerical identifier for reviewed beer

B. Preprocessing Data

As noted in II.5, the dataset used for the problem should describe a bipartite graph G. For this problem, we state the graph is comprised of disjoint sets B and U, which are the beers and users, respectively, whose information is stored in the dataset.

1) Summary Queries: To interpret the raw data into a bipartite graph, the data is organized into several groups using SQL's built-in GROUP BY queries. The following queries are written sequentially,

- Group the data by identifier of the reviewed beer (*beer_beerid*) and store the number of distinct reviewers who have reviewed that beer. This gives the set of beers B.
- Group the data by the profile name of the reviewer (*reviewer_profilename*) and store the number of distinct beers they have reviewed. This gives the set of users U.
- For each beer b and reviewer u, if u has reviewed b, return u's most recent of review of b. Return all five attributes reviewed. This gives the set of all edges in the bipartite graph (b,u) where $b \in B$ and $u \in U$, and where the ratings for each attribute comprise the edge weight tuple.

2) Minimum Degree Requirements: Due to the computational requirements of handling the entire set U and B we restrict the sets to product nodes with degree ≥ 30 and users with degree ≥ 20. We ended up with a total of 1,235,925 reviews (78%), 7,694 users (23%) and 7,136 products (11%).

3) Transformation into Tensor: Once the data is filtered through SQL we employ a 80-10-10 train, test, and validation split on the data. From this we create two tensors $\mathcal{A}$, the adjacency tensor consisting of beers and users and M the mask tensor which denotes whether a review is present or not.

C. Method

Given our two tensors $\mathcal{A}$ and $\mathcal{M}$, we employ masked CP Decomposition on these two tensors. We test three different models with various ranks. We use the following general algorithm:

Algorithm 1: Computing the Estimate $E(u, b)$

```
1: procedure DECOMP($\mathcal{A}, E', R$)    ▷ Given the adjacency tensor $\mathcal{A}$, validation set $E'$ and max rank $R$
2:     $r \leftarrow 1$
3: while $r \leq R$ do
       Initialize feature matrices $A_{(n)}$ randomly
4:     $\mathcal{T} \leftarrow Decomp(\mathcal{A}, \mathcal{M}, r, \mathbf{A})$
5:     Compute $\frac{1}{|E'|} \sum_{(u,b) \in E'} \|\mathcal{A}_{ub} - \mathcal{T}_{ub;r}\|^2$
6:     $R \leftarrow$ r if error is less than previous error.
7: end procedure
```

Our ranks are in the range $[1, 25]$ and Decomp function uses our Masked-CP decomposition. We use two versions of Decomp: MCP-Decomposition with SGD and MCP-Decompistion with PGD. We run until the relative iterative error is $\leq 10^{-3}$ or up to 500 iterations and sampled 12500 fibers. To achieve faster results, we use multiprocessing on our three separate models for each rank.

D. System Information

All experiments were performed on High-Ram Google Colab instances using python3. The tensor decomposition was written for this project but with basic tensor code using tensorly [6]. We also utilized numpy and the multiprocessing library to compute 3 decomposition at a time. We found any more would be constrained by the colab environment's 58 GB memory limitation.

E. Comparison Model

Here we introduce a simple comparison model that we will compare our tensor factorization method to. Let $U(u) \in \mathbb{R}^k$ be the average review user u gave in the training set and let $B(b)$ be the average review product b has in the training set. Then we let our estimate $\tilde{E}(u, b) = \lambda U(u) + (1 - \lambda) B(b)$ where λ is chosen from the validation set. It is clear that the comparison model is a tensor of rank at most $2k$.

V. Results and Discussion

In this section, we present our results from our tensor methods. We measure the error as a function of the rank of the decomposition. We compute the in-sample error E_{in} as well as the validation error E_{val} and take an average amongst our three models. This is shown in Figure 1 and Figure 2. In the figures, there is a similar trend in using both PGD and SGD. However, we find that PGD has a slight underhand producing an $E_{test} = 0.0906$. The best rank decomposition is of Rank 4. For rank > 4, the error starts to increase.

For our comparison model, we compute λ based on our validation set thus minimizing the validation error. Applying this value of λ, we get the following $E_{test} = 0.0534, E_{val} = 0.0615, E_{in} = 0.0613$. The comparison model here outperforms our tensor model by a factor ≈ 1.8.

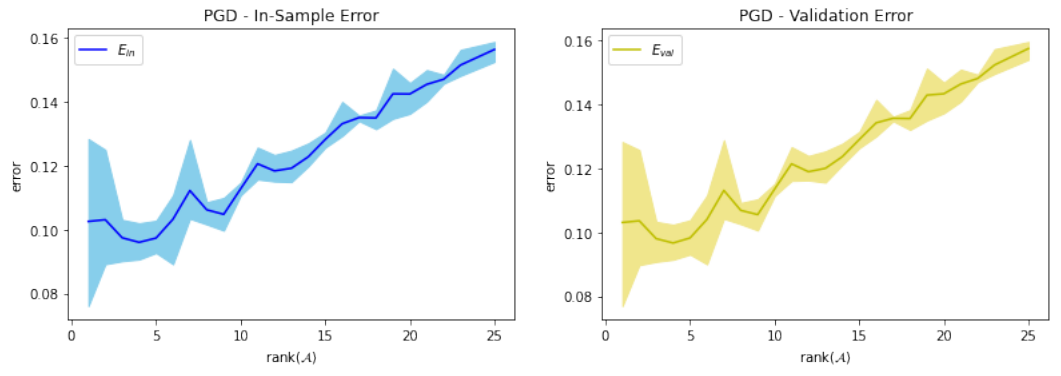

Fig. 1. Masked CP-Decomposition with Proximal Gradient Descent

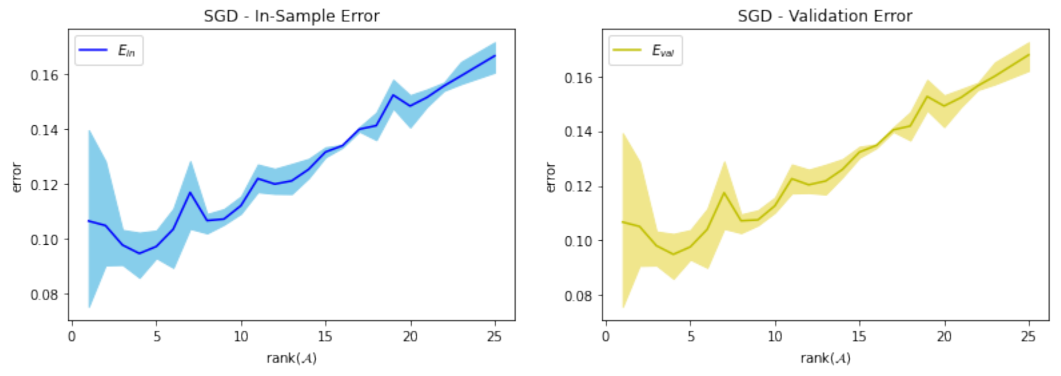

Fig. 2. Masked CP-Decomposition with Stochastic Gradient Descent

A. *Discussion*

Although there was success in modeling the problem, the in-sample error, E_{in}, is still relatively high. Due to limitations of computational resources, this was the best our model could perform. For our sparse data set with a large number of nodes, the tensor scales as a function of $O(MNk)$ and the cost per iteration of gradient descent is $O(F|\mathcal{F}_n|I_n)$. Other randomized CP-Decomposition algorithms could perform better such as second order methods such as those introduced in [5]. Additionally, if more computational resources were feasible, setting our threshold lower/running the algorithm for longer may provide for a smaller E_{in} yielding a better result.

VI. Conclusions

We have introduced a new model for predicting vector weighted edges in bipartite graphs based on computing the latent factors through CP decomposition. Our method is computationally expensive and suffers from a high in sample error when compared to a simple comparison model of rank ≤ 10. It is still possible with more computational resources, time or other algorithms that this method could out substantially out perform the comparison model.

Future work could comprise of non-uniform sampling methods or utilizing a non-variable sampling rate $|\mathcal{F}_n|$, for instance we could sample fibers corresponding to each node proportionally the degree of the node. We could also utilize the sparsity of both M and $\mathcal{X}$ to achieve a speedup in computing the decomposition. Investigating other decomposition such as one of the variants of the Tucker decomposition [2]. Exploring more ways to constrain each $A_{(n)}$ may also yield good generalization properties if E_{in} is made sufficiently small. Other dataset with not just vector edges more general tensor would also work with this frame work, though they would require an tensor of order 4 or higher and would be constrained by the computational difficulty.

References

[1] Y. Koren, R. Bell and C. Volinsky, ”Matrix Factorization Techniques for Recommender Systems,” in Computer, vol. 42, no. 8, pp. 30-37, Aug. 2009, doi: 10.1109/MC.2009.263.

[2] Kolda, Tamara G. and Bader, Brett W (2009). Tensor Decompositions and Applications. SIAM Review, 51(3), 455-500.

[3] Frolov, Evgeny, Huang, Ibrahim, Wai 2019]p2 Xiao Fu, Cheng Gao, Kejun Huang, Shahana Ibrahim, Hoi-To Wai, 2019 Block-Randomized Stochastic Proximal Gradient for Low-Rank Tensor Factorization.

[4] Yipeng Liu, Jiani Liu, Zhen Long, Ce Zhu, Yipeng Liu, Jiani Liu, Zhen Long, Ce Zhu, Tensor Decomposition, Tensor Computation for Data Analysis, 10.1007/978-3-030-74386-4, (19-57), (2022).

[5] Gittens, A., Aggour, K., & Yener, B. (2020). Adaptive Sketching for Fast and Convergent Canonical Polyadic Decomposition. In Proceedings of the 37th International Conference on Machine Learning (pp. 3566–3575). PMLR.

[6] Jean Kossaifi, Yannis Panagakis, Anima Anandkumar and Maja Pantic, TensorLy: Tensor Learning in Python, Journal of Machine Learning Research, Year: 2019, Volume: 20, Issue: 26, Pages: 16. http://jmlr.org/papers/v20/18-277.html.

[7] John Duchi, Elad Hazan, Yoram Singer (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research, 12(61), 2121-2159.