Collaborative Filtering

Arad Mohammadi, Andreas Tsouloupas, Hsiu-Chi Cheng, Omkar Zade, PICNIC Group
Department of Computer Science, ETH Zurich, Switzerland

***Abstract*—Collaborative Filtering (CF) is a common technique that is widely used by recommender systems in action. In short, CF is a method that makes automatic predictions or guesses about the opinions or interests of some people based on the opinions of other people who have similar interests with them. It is called CF because it can filter out items that a user likes based on collaboration of other users. In this paper we will discuss on the problem, the existing approaches to tackle it and our approach to solve the problem efficiently with a relative high accuracy.**

I. Introduction

Recommender systems are widely applied for many websites such as Netflix, Galaxus, Amazon, etc. Considering their user tastes, those systems suggest specific items to specific users which matches to them. Using such recommender systems has so many benefits for the company. Higher revenue, User satisfaction, Reducing time, Gaining new user are couple of these benefits.

CF is one of the most used approaches to the recommendation system problem. The benefit that this approaches has, is it does not depend on the relation between the items and or users. It tries to predict the outcome by analysing the past scores of a users given to items. Most CF methods using matrix factorization in order to understand the relation between the data better and improve their prediction.

In this paper first we discuss about the problem setting and then we describe the single models and methods that exist and try to compare them with each other, then we introduce our ensemble methods for current problem and the improvements that we have done and towards the end we talk about our results and the future works which can be applied to this problem.

II. Problem setting

We are given a matrix with 1176952 entries. These entries are the 10000 user ratings to 1000 different movies and the ratings are integer numbers between 1 to 5 which shows how much a specific user likes a specific movie. We should use this matrix also for tuning our different model's parameters so we devided this matrix into a training and a test(validation) set. Then we have to predict the missing enteries of the user-item matrix in order to recommend movies to users efficiently. For final evaluation between models, root mean squared error(RMSE) is used. RMSE calculate the root of average distance between each predicted rating and true rating. The formula will be

$$RMSE = \sqrt{(\frac{1}{n})\sum_{i=1}^{n}(r_{i,truth} - r_{i,predicted})^2}$$

which $r_{i,truth}$ is the truth rating, $r_{i,prediction}$ is the predicted rating and n is the number of items in the test database. The final leaderboard is calculated with approximately 50% of the test data which is the public test set. The other 50% is the private test set so the final leaderboard scores might be different with the scores we stated in our paper.

III. Single Models and Methods

In our CF approaches, we focused on matrix factorization based models. In this section, we will present seven single algorithms we used for our recommender systems.

A. Overview

The idea behind matrix factorization based models is that preferences of user u for item i can be estimated by a few hidden factors. Here we describe the association between user u and the factors as $p_u \in \mathbb{R}^k$, while that between item i and the factors as $q_i \in \mathbb{R}^k$. Under this structure, the estimation of a rating of user u for item i is modelled by the dot product of these two vectors p_u and q_i.

$$\hat{r}_{ui} = p_u^T q_i = \Sigma_{t=1}^{k} p_{ut} q_{ti} \tag{1}$$

Suppose there are n users and m items, we can assemble all those user factors as a matrix $P \in \mathbb{R}^{k \times n}$ and item factors as $Q \in \mathbb{R}^{k \times m}$. Once we calculate the dot product of P and Q, we are able to estimate all the user-item interactions by a matrix $\hat{A} \in \mathbb{R}^{n \times m}$ where

$$\hat{A} = P^T Q. \tag{2}$$

Our goal of matrix factorization based methods was to figure out the matrices P and Q from limited observed ratings and make estimations of unknown ratings.

B. Missing Values

In our case, we have 10,000 users and 1,000 items, which forms a user-item interaction matrix with 10,000,000 values. There, however, are only 1,176,952 observed values which is just 11.77%, with two decimals accuracy, of the whole matrix. In order to model our recommender systems, one issue is to deal with missing values in this sparse matrix. Here, we made use of following two methods:

- Missing value imputations: we imputed zero to those missing ratings.
- Error calculations: we only calculated errors between observed values and their estimated values.

C. Singular Value Decomposition (SVD)

SVD [1] is a popular method in linear algebra for matrix factorization. SVD factorizes a given matrix $A \in \mathbb{R}^{n\times m}$ into two orthogonal matrices $U \in \mathbb{R}^{n\times n}$ and $V \in \mathbb{R}^{m\times m}$, and a rectangular diagonal matrix $\Sigma \in \mathbb{R}^{n\times m}$, such that

$$A = U^T \Sigma V,\ \Sigma = diag(\sigma_1, ..., \sigma_{\min\{n,m\}}) \tag{3}$$

where $\sigma_i \geq \sigma_{i+1} \geq 0$. From the description of III-A, our goal is to find a low-rank approximation $\hat{A}$ such that $rank(\hat{A}) \leq k$. Eckart-Young Theorem [2] shows that pruning the first k singular values in the SVD representation results in the optimal rank k approximation of a matrix. We thus only utilized the largest k values in Σ and first k rows of U and V. We can rewrite our estimation matrix into

$$\hat{A} = U_k^T \Sigma_k V_k \tag{4}$$
$$= (U_k^T \Sigma_k^{\frac{1}{2}})(\Sigma_k^{\frac{1}{2}} V_k) \tag{5}$$
$$= P^T Q \tag{6}$$

where $\Sigma_k = (\sigma_1, ..., \sigma_k) \in \mathbb{R}^{k\times k}$, $U_k = (u_1, ..., u_k) \in \mathbb{R}^{k\times n}$, u_i is the i^{th} row in U; $V_k = (v_1, ..., v_k) \in \mathbb{R}^{k\times m}$, v_j is the j^{th} row in V.

D. Alternating Least Square (ALS)

One of the common approaches to CF is ALS [3]. ALS iteratively optimizes the objective function

$$\ell(P, Q) = \frac{1}{2}||\Pi_\Omega(A - P^T Q)||_F^2 + \frac{\lambda}{2}(||P||_F^2 + ||Q||_F^2) \tag{7}$$

Here $||\cdot||_F$ is the Frobenius norm, and Ω is a $n \times m$ index matrix W with values in $\{0, 1\}$, where $w_{ij} = 1$ if A_{ij} is observed. ALS monotonically improves the objective and that converges to a fixed point but there is no guarantee that it will converge to the optimal. This algorithm can be expressed in mathematics by

Algorithm 1 ALS

Input: Matrix $A \in \mathbb{R}^{n\times m}$, truncation $k < \min\{n, m\}$, initial guesses latent factors $P^0 \in \mathbb{R}^{k\times n}$ and $Q^0 \in \mathbb{R}^{k\times m}$

1: $t \leftarrow 0$
2: **while** not convergent **do**
3: $\quad Q^{t+1} \leftarrow \text{argmin}_Q\, \ell(P^t, Q)$
4: $\quad P^{t+1} \leftarrow \text{argmin}_P\, \ell(P, Q^{t+1})$
5: **end while**
6: **return** $\hat{A}$ as in Equation 2

E. Global biases

The idea of global biases method is that each user and each item have a specific bias towards ratings. We calculated those biases first, removed them, and then extracted hidden factors. Afterwards we added back those biases for specific users and items.

Algorithm 2 Global biases

Input: User-item interaction matrix $A \in \mathbb{R}^{n\times m}$, index matrix $W \in \mathbb{R}^{n\times m}$, scalar $lambda$

1: Calculate rating mean of user u: $\mu_{(u,.)} = \frac{\Sigma_{i=1}^m w_{ui} a_{ui}}{\Sigma_{i=1}^m w_{ui}}$, rating mean of item i: $\mu_{(.,i)} = \frac{\Sigma_{u=1}^n w_{ui} a_{ui}}{\Sigma_{u=1}^n w_{ui}}$, rating bias of user u: $b_{(u,.)} = \mu_{(u,.)} - \frac{\Sigma_{v=1}^n \mu_{(v,.)}}{n}$, and rating bias of item i: $b_{(.,i)} = \mu_{(.,i)} - \frac{\Sigma_{j=1}^m \mu_{(.,j)}}{m}$
2: $B_u \leftarrow (b_{(1,.)}, ..., b_{(n,.)})$
3: $B_i \leftarrow (b_{(.,1)}, ..., b_{(.,m)})$
4: **while** not convergent **do**
5: $\quad$ Calculate global rating mean $\mu = \frac{\Sigma_{u=1}^n \Sigma_{i=1}^m w_{ui} a_{ui}}{\Sigma_{u=1}^n \Sigma_{i=1}^m w_{ui}}$
6: $\quad$ Make use of ALS to optimize the global bias μ with obejective funcion

$$\ell(B_u, B_i) = \frac{1}{2}||\Pi_\Omega(A - \mu - (B_u \dot{1}^{1\times m})^T - B_i 1^{1\times n})||_F^2 + \frac{\lambda}{2}(||B_u||_F^2 + ||B_v||_F^2)$$

7: **end while**
8: **return** $\hat{A}$ with $\hat{a}_{ij} = \mu + b_{(u,.)} + b_{(.,i)}$

F. Improved Regularized SVD

Inspired by gradient descent, we modified the global biases method by updating variables with gradient descent. There are two ways to initialize rating bias of user u and item i, denoted by $b_{(u,.)}$ and $b_{(.,i)}$ respectively.

- Zero vectors: initialize them with zero vectors.
- User-item biases: initialize them with $b_{(u,.)} = \mu_{(u,.)} - \frac{\Sigma_{v=1}^n \mu_{(v,.)}}{n}$ and $b_{(.,i)} = \mu_{(.,i)} - \frac{\Sigma_{j=1}^m \mu_{(.,j)}}{m}$

Algorithm 3 Improved Regularized SVD

Input: User-item interaction matrix $A \in \mathbb{R}^{n\times m}$, index matrix $W \in \mathbb{R}^{n\times m}$, scalar λ_1, λ_2, and η

1: Calculate rating mean of user u: $\mu_{(u,.)} = \frac{\Sigma_{i=1}^{m} w_{ui}a_{ui}}{\Sigma_{i=1}^{m} w_{ui}}$, rating mean of item i: $\mu_{(.,i)} = \frac{\Sigma_{u=1}^{n} w_{ui}a_{ui}}{\Sigma_{u=1}^{n} w_{ui}}$, rating bias of user u: $b_{(u,.)} = \mu_{(u,.)} - \frac{\Sigma_{v=1}^{n}\mu_{(v,.)}}{n}$, and rating bias of item i: $b_{(.,i)} = \mu_{(.,i)} - \frac{\Sigma_{j=1}^{m}\mu_{(.,j)}}{m}$
2: $B_u \leftarrow (b_{(1,.)}, ..., b_{(n,.)})$
3: $B_i \leftarrow (b_{(.,1)}, ..., b_{(.,m)})$
4: Initialize $U \in \mathbb{R}^{k\times n}$ and $V \in \mathbb{R}^{k\times m}$ by drawing numbers from a normal distribution with mean= 0 and std=$\frac{1}{k}$
5: **while** not convergent **do**
6: Calculate global rating mean $\mu = \frac{\Sigma_{u=1}^{n}\Sigma_{i=1}^{m} w_{ui}a_{ui}}{\Sigma_{u=1}^{n}\Sigma_{i=1}^{m} w_{ui}}$
7: Calculate obejective funcion

$$\begin{aligned}\ell(U,V,B_u,B_i) =& \frac{1}{2}||A - ((\mu + B_u 1^{1\times m})^T \\ &+ B_i 1^{1\times n} + U^T V)||_F^2 \\ &+ \frac{\lambda_1}{2}(||B_u||_F^2 + ||B_v||_F^2) \\ &+ \frac{\lambda_2}{2}(||U||_F^2 + ||V||_F^2)\end{aligned}$$

8: Update a_{ij} with gradient descent
9: $u_{ik} \leftarrow u_{ik} + \eta[a_{ij} - (\mu + b_{(i,.)} + b_{(.,j)} + u_i^T v_j)v_{jk} - \lambda_1 u_{ik}]$
$v_{jk} \leftarrow v_{jk} + \eta[a_{ij} - (\mu + b_{(i,.)} + b_{(.,j)} + u_i^T v_j)u_{ik} - \lambda_1 v_{jk}]$
$b_{(i,.)} \leftarrow b_{(i,.)} + \eta[a_{ij} - (\mu + b_{(i,.)} + b_{(.,j)} + u_i^T v_j) - \lambda_2 b_{(i,.)}]$
$b_{(.,j)} \leftarrow b_{(.,j)} + \eta[a_{ij} - (\mu + b_{(i,.)} + b_{(.,j)} + u_i^T v_j) - \lambda_2 b_{(.,j)}]$
10: **end while**
11: **return** $\hat{A}$ with $\hat{a}_{ij} = \mu + b_{(u,.)} + b_{(.,i)} + u_i^T v_j$

G. Singular Value Projection (SVP)

SVP [4] is a projected gradient descent method. For a given matrix A, SVP iteratively makes an orthogonal projection onto a set of k-rank matrices by

$$A^0 = 0,\ A^{t+1} = [A^t + \eta\Pi_\Omega(A - A^t)]_k,\ \eta > 0,$$

where the projection to rank k matrices is done by SVD. Before we did SVP, we also normalized our data by Z-score first, where mean and standard deviation were only calculated by observed data.

H. Singualar Value Thresholding (SVT) / Nuclear norm relaxation

SVT [5] has been successfully used in many low-rank optimization problems. It is an algorithm to iteratively minimize the nuclear norm of a matrix, subject to certain types of constraints, by the algorithm as follows.

Algorithm 4 SVT

Input: Matrix $A \in \mathbb{R}^{n\times m}$, index matrix $W \in \mathbb{R}^{n\times m}$, $\tau \in \mathbb{R}^+$, $T \in \mathbb{Z}^+$

1: $\hat{A} \leftarrow A$
2: **for** t in $1, ..., T$ **do**
3: SVD decomposition: $\hat{A} \leftarrow U^T diag(\sigma_i)V$
4: $\hat{A} \leftarrow U^T diag(\sigma_i - \tau)_+ V$
5: **end for**
6: **for** i in $1, ..., n$ **do**
7: **for** j in $1, ..., m$ **do**
8: **if** w_{ij} = 1 **then**
9: $\hat{a}_{ij} = a_{ij}$
10: **end if**
11: **end for**
12: **end for**
13: **return** $\hat{A}$

I. Soft Impute

One of the available methods for matrix completion is Soft Impute [6]. Soft Impute method itself uses SVD with a simple trick. It was state of the art for a period of time.

Soft Impute solves the following problem for a matrix A with missing entries:

$$\min ||A - M||_F^2 + \lambda||M||_*. \tag{8}$$

Here $||M||_*$ is the nuclear norm of M (sum of singular values) For this problem first we used the Bisacler library in order to normalize rows and columns. Then we used the Softimpute for 10 iterations and scaled the data back into ratings 1 through 5. After that we saw the MAE score and compare it with other methods.

On of the tricks that we used here was adding data iteratively. When we scaled back the data in to 1 through 5, most of the predictions were not integer number, so we rounded the prediction to the closest integer (distance less than 0.05). These prediction were the most confident predictions. So for next time, we added this data to our matrix and our matrix has less missing values this time, so it helps to predict the missing values better. But we have to pay attention to overfitting, so we tune the hyperparameters and decided to add this data just 2 times to the original matrix.

IV. Ensemble methods

So far, we have presented various models which yield satisfactory performance on the dataset. A natural question to ask is whether we can combine the predictive power of these models. *Ensembling* is a simple technique to achieve this. We used a variant of ensembling (proposed in [7] but not implemented by the author due to computational

constraints at the time), commonly referred to as *Blending*, to combine our models. Assuming we have n models and the reconstructed matrix by the i-th model is denoted by $\hat{A}^i$, the final prediction is the weighted linear combination

$$\hat{A} = w_0 + \sum_{i=1}^{n} w_i \hat{A}^i \tag{9}$$

where $\hat{A}^i$ are fixed, and we learn the parameters w_i for $i = 0, 1, ..., n$. This reduces to solving a least-squares problem. However, if we use the training data on which $\hat{A}^i$ are trained, the ensemble is prone to overfitting. Hence, the following strategy is employed:

1) Split the dataset into training and a holdout set (e.g. 90-10 split)
2) Train $\hat{A}^i$ on the training set, learn w_i on the holdout set

We tested this method and it beats all the individual models, however we are using only 90% of the available data to build $\hat{A}^i$. To utilize the entire labeled dataset in order to further improve performance, K-fold cross validation can be used. The ensemble algorithm is then:

Algorithm 5 K-fold_Ensemble($\hat{A}^1, \ldots, \hat{A}^n$)

Split the data into K folds of equal size
for j in $1...K$ **do**
  Keep the jth fold as holdout data
  for i in $1...n$ **do**
    Train $\hat{A}^i$ on the remaining $K-1$ folds
    Let X_{ij} be the prediction vector of $\hat{A}^i$ on the jth fold (the holdout set)
  end for
  Solve the linear system

$$y_j = w_{0j} + \sum_{i=1}^{n} w_{ij} X_{ij}$$

  where y_j are the true labels of the jth holdout set
end for
Compute $w_i = \sum_{j=1}^{K} w_{ij}/K$
Re-train all $\hat{A}^i$s on the entire dataset
return $\hat{A}$ as in Equation 9

A. Models

1) Baseline method: SVD + ALS
2) Global biases
3) Improved Regularized SVD
4) SVProjection
5) Nuclear norm relaxation / SVT
6) Regularized SVD

References

[1] V. Kelma and A. Laub, "The singular value decomposition: Its computation and some applications," *IEEE Transactions on Automatic Control*, vol. 25, no. 2, pp. 164–176, 1980.

[2] C. Eckar and G. Young, "The approximation of one matrix by another of low rank," *Psychometrika*, vol. 1, no. 3, pp. 211–218, 1936.

[3] R. B. Y. Koren and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2017.

[4] P. Jain, R. Meka, and I. Dhillon, "Guaranteed rank minimization via singular value projection," *Advances in Neural Information Processing Systems*, vol. 23, 2010.

[5] J.-F. Cai, E. J. Candès, and Z. Shen, "A singular value thresholding algorithm for matrix completion," *SIAM Journal on optimization*, vol. 20, no. 4, pp. 1956–1982, 2010.

[6] T. Hastie, R. Mazumder, J. Lee, and R. Zadeh, "Matrix completion and low-rank svd via fast alternating least squares," 2014.

[7] A. Paterek, "Improving regularized singular value decomposition for collaborative filtering," in *Proceedings of KDD cup and workshop*, vol. 2007, 2007, pp. 5–8.