Project for 236799 – Collaborative Filtering

Tal Skverer, ID:312489560, Email: talsk@campus.technion.ac.il

24.8.2020

1. Introduction

In many aspects of our lives, we utilize *recommendation systems* as a way to discover new content that is relevant to us and that we are interested in.

Thus, these systems must provide accurate recommendations that will, with high probability, interest the users, and do so quickly, both in the sense of being able to adapt to new information about certain users or content, and also produce actual recommendations on the fly.

Collaborative Filtering (CF) is a specific technique used by recommendation systems, on which this project will focus.

Abstractly, it is a method of making predictions about interests ("filtering") of a user based on a collected information about many other users ("collaborative").
Concretely, we are given ratings for N users and M items. A certain value for user u and item i if user u interacted with item i in some way, either explicitly (a user rated a book) or implicitly (a user visited some store's page).
Sometimes, we also have *auxiliary information*, containing extra information about users (gender, age, location) and items (type, popularity).

Then, given some specific user and item, the method uses said data to predict whether the user will be interested in the item, and how much.

In this project, we mainly review four literature papers discussing specific CF methods.
One that surveys many "traditional" methods, and three that present new, deep-learning-based approaches.
These papers appear first in the list of references, respectively receiving the numbers [1],[2],[3],[4] and were underlined when referenced throughout the project for clarity.

We start by detailing an overview of CF methods, split into three distinct types. During this process, we take some representative methods (usually from the reviewed papers) and further detail them. We also describe the datasets used in the papers.

We then discuss shortcomings of a recommendation system based on CF techniques, and detail known solutions to these problems.

We continue by presenting an architecture of a full recommendation system based on one of the representative methods.

We finish by conducting an experimental evaluation, for which we implement two chosen representative methods, and reproduce the results presented in their original papers, on the original datasets. We further experiment by taking a new dataset, of a domain different to these the papers use, and check how well the methods perform.

2. Literature review

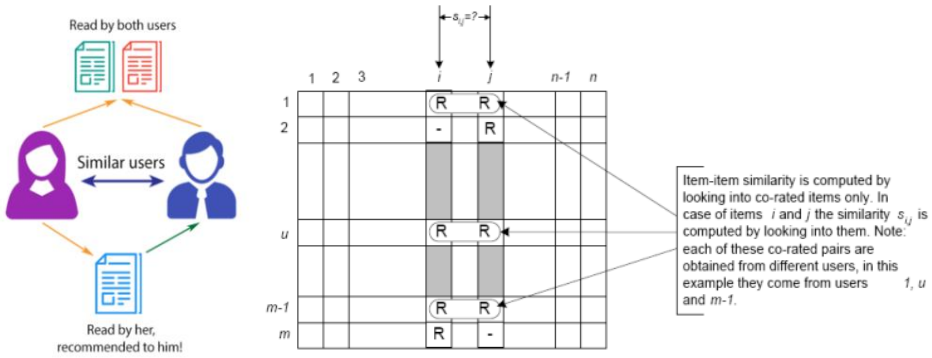
As defined by *Ricci et al.* [5], CF methods produce user specific recommendations of items based on patterns of ratings or usage without need for exogenous information about either item or users.

These methods operate within a system of N users and M items, and are sequentially given an $N \times M$ *rating matrix* r, that for a user u and an item i, a value $r_{u,i}$ indicates the preference by user u of item i, where higher values mean stronger preference.
A cell with no values for user-item pairs means that they have not yet interacted.

Some methods also use auxiliary information provided about the users and items. In this case, we call this information item or user *features*, and denote the number of features per users as P, and the number of features per item as Q. Hence, in such case we have two additional matrices, X of size $P \times N$ and Y of size $Q \times N$, representing auxiliary information about users and items, respectively.

A distinction made by *Linden et al.* [6] points at two methodologies of CF:

User-based, where the method looks for users who had similar rating patterns and uses these "neighbors" to make a prediction for an active user (figure 1).
Item-based systems (also called item-to-item), on the other hand, focus on finding similar items for each item that an active user had interacted with, and use these to make predictions (figure 2).



Figures 1,2. User-based collaborative filtering[1], and item-based similarity computation diagram[2].

Then, in *A Survey of Collaborative Filtering Techniques*, [4] *Su et al.* detail three different types (or categories) of collaborative filtering methods: **Memory-based**, **Model-based** and **Hybrid methods**.

We shortly overview these categories and dive deeper into one representative method.

I. Memory-based

These algorithms use the entire or sample of the provided information to generate a prediction. They use this data to group people with similar interests (neighbors), then use this grouping to produce a prediction about an active user. Thus, they are also dubbed neighborhood-based algorithms.

The neighborhood-based algorithms can be generalized [7] to the following steps: calculate similarity $w_{u,v}$ between uses u and v. Then, a prediction can be made by aggregating these similarities values in a defined way, based on the required task.

---

[1] From "Brief on Recommender Systems", in blog post by Sanket Doshi.
[2] From "Recommender Systems" exercise by Carleton College.

For the representative method of this type, we take a simple, user-based neighborhood algorithm that uses correlation-based similarity calculation, and prediction computation based on a weighted summation.

We take inspiration from *Resnick et al.* [8] and use *Pearson correlation* to compute similarities between users u, v:

$$w_{u,v} = \frac{\sum_{i \in I_{u,v}} (r_{u,i} - \bar{r}_{u,I_{u,v}})(r_{v,i} - \bar{r}_{v,I_{u,v}})}{\sqrt{\sum_{i \in I_{u,v}} (r_{u,i} - \bar{r}_{u,I_{u,v}})^2} \sqrt{\sum_{i \in I_{u,v}} (r_{v,i} - \bar{r}_{v,I_{u,v}})^2}}$$

Where $I_{u,v} \subseteq [M]$ is the subset of items both users u, v rated, and $\bar{r}_{u,I_{u,v}}$ is defined as $\frac{\sum_{i \in I_{u,v}} r_{u,i}}{|I_{u,v}|}$ - the average rating by user u of the items in $I_{u,v}$.

We continue by picking a prediction computation based on the *weighted sum of other users' ratings*. Hence, to make a prediction for an active user u on a specific item i, we take a weighted average of all the ratings on that item:

$$\tilde{r}_{u,i} = \bar{r}_u + \frac{\sum_{v \in V_i} (r_{v,i} - \bar{r}_v) \cdot w_{u,v}}{\sum_{v \in V_i} |w_{u,v}|}$$

Where $V_i \subseteq [N]$ is the subset of all users who have rated item i, and $\bar{r}_v$ is the average ratings of user v, excluding item i. In this formula, we can clearly see how the similarity between users affect the predicated rating – the more a user v is similar to u (correlation-wise, in our case) the larger their ratings will impact the final, predicted rating.

II. Model-based

These algorithms include all methods that produce a model by learning complex patterns based on the training rating matrix. This type includes most of the state-of-the-art algorithms, as they are usually developed to overcome unwanted behaviors of past, memory-based algorithms, and thus became prevalent in today's collaborative filtering techniques [9].

Most traditional machine-learning algorithms can be used to create such model-based solutions: Bayesian classifiers, clustering models, decision trees, rule-based methods and recently - neural networks, specifically those with deep architectures [10].
Usually, categorical user ratings are solved by using classification models, and numerical ratings by regression models.

Su et al. [4] detail several model-based techniques – Simple Bayesian classifiers, Extended logistic regression (ELR) models, clustering algorithms, regression-based algorithms, Markov decision processes and latent-semantic models.
They also briefly discuss *Matrix factorization* (MF) algorithms, which have shown promising results regarding large and sparse CF tasks [11].
Due to the importance of these issues, and mostly due to the Netflix Prize[3] challenge, in the time since the paper was published this type of algorithms had been thoroughly researched and their expected results were verified. Thus, a basic MF algorithm is chosen as a representative for this type.

The idea behind MF for CF, as was originally suggested by Simon Funk [12], is to map the user-item rating matrix to a joint latent space of lower dimensionality D (the number of *latent factors*), such that interactions are modeled as inner products in that space [13].
This process is done by factorizing said matrix to a product of two matrices, H, which has a row for each user, and W, which has a column for each item.
The predicted ratings is the product of these two matrices, or more specifically, for a user u

[3] An open competition for the best CF algorithm held by Netflix in 2009.

and item i, we can calculate the predicated rating by taking the inner product of the relevant column and row,

$$\tilde{r}_{u,i} = (H^T)_u \cdot W_i$$

Thus, this model minimizes some objective function on the set of known user-item ratings K. One, well-received option is to use Probabilistic Matrix Factorization (PMF), which assumes a probabilistic model with Gaussian observation noise and priors on the latent factors [14]. Maximizing the log posterior on the parameters leads to the following objective function:

$$\underset{H,W}{\arg\min} \sum_{(u,i)\in K} \left(r_{u,i} - \tilde{r}_{u,i}\right)^2 = \underset{H,W}{\arg\min} \sum_{(u,i)\in K} \left(r_{u,i} - (H^T)_u W_i\right)^2 \quad (1)$$

We notice that the expressive power of this model is directly influenced by the number of latent factors, with more factors improving the general recommendation quality.
Of course, too many factors may cause the model to overfit to the example set. To avoid overfitting, we can add regularization term on the matrices H, W, so the objective function (1) changes to:

$$\underset{H,W}{\arg\min} \sum_{(u,i)\in K} \left(r_{u,i} - (H^T)_u W_i\right)^2 + \lambda(\|H\|^2 + \|W\|^2) \quad (2)$$

With λ being a regularization parameter, and the matrix norms used depend on the specific CF task.

A recent trend is to incorporate deep neural networks with MF algorithms.
In *Neural Collaborative Filtering*, [1] *He et al.* presents a general framework for Neural-based CF algorithms, dubbed NCF.

In NCF, the bottom layer is some representation of the user and item vectors, fully connected to an embedding layer, which results in the factorized, latent vectors.
Then, these latent representations are fed into a multi-layer neural architecture. This architecture can contain any number of layers, each customized, in a sense of their connectivity and activation function, to discover certain latent structures of user-item interactions (similar to how Convolutional Neural Networks work on images).
The last layer's activation function must map to (categorial or numerical) rating, to produce a prediction.

Learning the model's parameter can be achieved by minimizing an objective function like the one previously shown. Using the notation $\tilde{r}_{u,i}$ for the output of the network:

$$\underset{H,W}{\arg\min} \sum_{(u,i)\in K\cup K^-} \alpha_{u,i}\left(r_{u,i} - \tilde{r}_{u,i}\right)^2$$

K is defined as above, K^- is the set of all negative instances, which can be all (or a sample) of unobserved interactions, and $\alpha_{u,i}$ is a hyper-parameter denoting the weight of training instances (These additions are similar to those suggested in the paper).

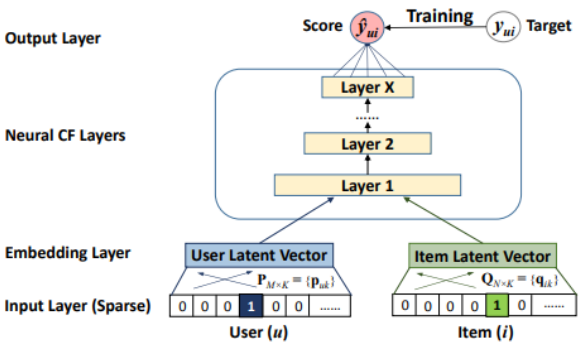


Figure 2. Neural CF framework. The matrices P and Q are H and W respectively in our notation.

Under this framework, it is easy to represent the original MF algorithm: A single layer operating an element-wise function between the latent vectors, and the identity activation function.

It is then natural to try a standard multi-layer perceptron neural network – this will endow the model a large level of non-linearity to learn complex interactions between the factorized, latent vectors, instead of the previous, fixed element-wise product of them.

In the reviewed paper [1], the MLP network was defined with L layers, with ReLU activation function between layers, as it was proven to be non-saturated, encourages sparse activations and generally works well for sparse data [15], whereas the output layer uses the Sigmoid function. These choices were also verified empirically.

The layers themselves follow a tower structure, where the higher layers include fewer neurons. The premise is that this way, the higher layers may learn more abstractive features of the data, again, like CNNs [16].
Finally, the paper suggested a fusion between the traditional MF, and the new, MLP-based network under the suggested framework, denoting the name *Neural Matrix Factorizing* (*NeuMF*), hoping to get the best-of-both-worlds out of the methods.

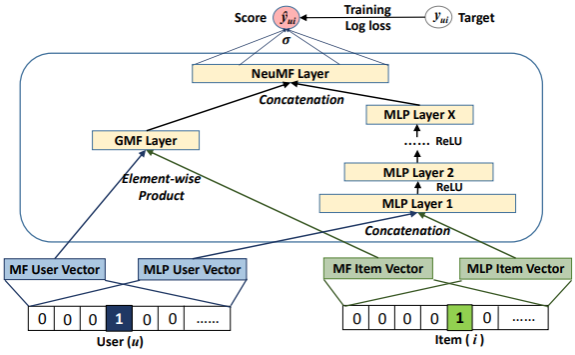


Figure 3. Visualization of the NeuMF model

III. Hybrid methods

These combine previously discussed CF algorithms with other, *information-based* recommendation techniques. Typically, these other systems make recommendations by analyzing auxiliary information about the users or items.
Some examples are *content-based* systems, that analyze textual information and finding regularities in it, *demographic-based* systems, which use user profile information, and *utility-based/knowledge-based* which process information about what satisfies the user's needs. These systems then use heuristic or classification approaches to make recommendations.

According to *Su et al.*, creating a hybrid model is done by either adding information-extracted characteristics to CF models, adding CF-extracted characteristics to information-based models, making predictions based on multiple CF and information-based models, or combining different CF models.

The paper gives an overview of few different implementations of hybrid systems.
One example is a *content-boosted* CF algorithm, that uses Naïve Bayes classifier to complete the missing values in the rating matrix, creating a "pseudo rating matrix". Then, using this matrix for a CF method (such as MF) it can give a final recommendation.
Another example is to combine different recommendation techniques, giving a different weight to each of them for the final prediction, such as majority voting, average predicated rating, or adapting the weight using confidence levels of the methods.

Notation-wise, we recall we denoted a $P \times N$ matrix X as the user-feature matrix, that contains a row of length P for every user, which contains all auxiliary information about this user. Similarly, we denoted a $Q \times M$ matrix Y as the item-feature matrix, that contains all auxiliary information about all items.

Since two of the papers we survey use auxiliary information with a CF algorithm to create a hybrid method in a similar fashion, we will review them as the representatives for this type of methods.

In *Deep Collaborating Filtering via Marginalized Denoising Auto-encoder,* [2] *Li et al.* suggest incorporating marginalized denoising autoencoder (mDA) as a deep-learning approach to learn latent representation of the user or item features.

Autoencoders [17] are a specific type of neural network, which learn some latent representation of the input by a process of encoding to a smaller dimension and then decoding to recreate the original input. Specifically, denoising autoencoder work with (usually) randomly corrupted input.
Multiple autoencoders can be stacked to construct a deep network and have shown promising result in learning higher level representations [18]. A variant which excels in coping with large number of parameters is the marginalized denoising autoencoders, that has a closed-form solution for the parameters and thus is much less computationally expensive [19].

The process of learning with mDA is done by minimizing the squared loss which gives us the following objective function:

$$\arg\min_{A} \frac{1}{2ck} \sum_{j=1}^{c} \sum_{i=1}^{k} \left\| x_i - A(\tilde{x}_i)_j \right\|^2$$

where x_i indicate the i^{th} term of the input set (user i), and the process is done c times, with $(\tilde{x}_i)_j$ indicating the j^{th} corrupted version of the relevant term. Under this function, the network effectively learns some matrix A, a mapping that best reconstructs the input. Using this mapping, the objective function can also be written as

$$\arg\min_{A} \left\| \bar{X} - A\tilde{X} \right\|_F^2$$

where $\bar{X}$ is a matrix containing X c times, and $\tilde{X}$ is the corresponding corrupted versions. This is effectively a least-squares problem, for which a closed-form solution is known using pseudo-inverse.

Li et al. present a small change to the previously-discussed object function (2) of matrix factorization method to create a general **hybrid** model by adding two new functions $\mathcal{L}(X, H), \mathcal{L}(Y, W)$ that connect the user or item feature matrices with the matrices

representing the latent factors. They denote this framework *Deep Collaborative Filtering* (DCF).

$$\arg\min_{H,W} \sum_{(u,i)\in K} \left(r_{u,i} - (H^T)_u W_i\right)^2 + \beta(\|H\|^2 + \|W\|^2) + \gamma\mathcal{L}(X,H) + \delta\mathcal{L}(Y,W) \quad (3)$$

With β, γ, δ set as hyper-parameters.

They then develop a specific hybrid CF model using mDA. The functions that connect the user and item features with the latent factors are defined as follows:

$$\mathcal{L}(X,H) = \left\|\bar{X} - A_1\tilde{X}\right\|_F^2 + \lambda_1\|P_1H^T - A_1X\|_F^2$$
$$\mathcal{L}(Y,W) = \left\|\bar{Y} - A_2\tilde{Y}\right\|_F^2 + \lambda_1\|P_2W^T - A_2Y\|_F^2$$

A_1,A_2 are the reconstruction mapping as defined in the mDA formalization above, P_1, P_2 are projection matrices, and the objective function (3) is set to minimize over H, W, A_1, A_2, P_1 and P_2.

The first term in these loss functions is simply the learning process in mDA once for items and once for users, and the second term connects the hidden layer features (A_1X/A_2Y) with the respective latent matrix (H/W). The learnable projections P_1, P_2 are used to map the latent factors to the feature space.

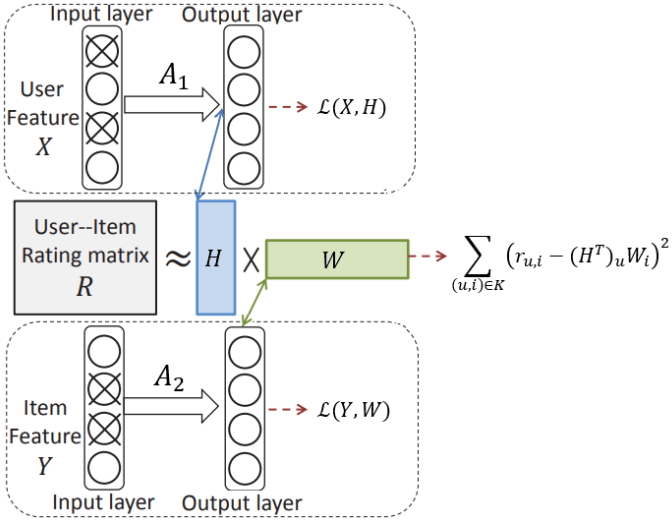


Figure 4. Illustration of the PMF + mDA learning process (adapted from the original paper)

The objective function (3) is not jointly convex, thus *Li et al.* suggest optimizing it iteratively, fixing different variables every step.

Additionally, a similar approach using "stacked" auto-encoders was briefly discussed – by having multiple layers of mDA and taking the middle layer's mapping A in the losses defined above.

This approach is further expanded in *Collaborative Deep Learning for Recommender Systems* [3]. In this paper, *Wang et al.* formulate a method using stacked denoising autoencoder (SDAE) to learn a latent representation for item features that is combined as a hybrid model with an MF algorithm.

In this approach, denoted *Collaborative Deep Learning* (CDL), the encoder layers of the SDAE shrink in size gradually up to the middle layer. The premise is that the latent representation will be easier to learn, instead of immediately encoding to a small dimension.

Then, the small, $L/2$ layer (here L is the total number of layers) serves as a bridge between the ratings and the content information.

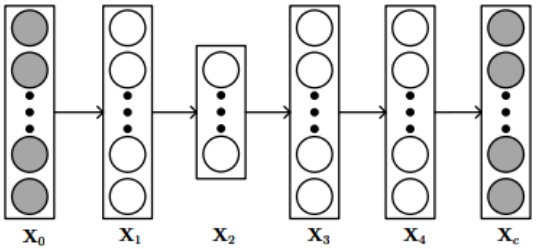


Figure 5. An SDAE with 4 layers.
In this case, the 2nd layer is used as the latent representation.

A generative process is defined for CDL, in which the SDAE learns from Gaussian noise on the inputs versus the clean inputs.

Then, this minimization of (2) combined with the output of the SDAE's $\frac{L}{2}$ layer is used to create the matrices H and W.

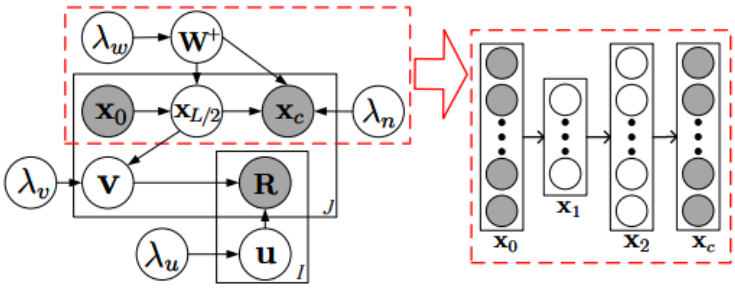


Figure 6. Graphical view of the CDL model with 2 layers ($x_{L/2}$ translates x_1), taken from the paper.
In their notation, the λs are hyper-parameters, W^+ are the learnt mapping of the SDAE
v is the item latent factors matrix W, and u is the user latent factors matrix H.

IV. Datasets

Few publicly accessible datasets were used in the reviewed papers. An overview of their characteristics and links (if available) can be seen in table 1.

MovieLens – movie rating from the same-named website containing two benchmark datasets, one with 6,000 users and one million ratings, with each user having at least 20 and the other with 900 users and 100 thousand ratings. Having original ratings from 1 to 5, this dataset contains explicit feedback. These datasets consist of some user information, such as age, gender, and occupation, as well as movie genre as item information.

Pinterest – constructed implicit feedback dataset containing whether a user had pinned specific item. Since over 20% of users have only one pin, the dataset were filtered to include only users with at least 20, resulting in 55,187 users and over one and a half million interactions.

Book-Crossing – contains over 1 million ratings for over 270 thousands books by 270 thousand users. This is also explicit feedback, with ratings ranging from 0 to 10. Some item and user information is also provided.

Advertising – implicit feedback data containing 700 advertisements and 400 thousand users, and whether a user interacted with an ad. Additionally, it includes user information, and meta information about the ads.

Netflix - contains two datasets, one officially released as part of the Netflix challenge, containing ratings and movie titles, and the other collected by *Wang et al.* [3] from IMDB, contains only positive ratings and users with 3 or more ratings, and was combined with the former for a total of over 400 thousand users, 9,000 movies and about 15 million ratings.

Citeulike – contains two datasets labeled 'a' and 't' which were collected independently of each other. They include explicit feedback – articles and users who collected them. The first contains 5,000 users and 17 thousand items, and the second 8,000 users and 26,000 items.

Dataset	N. interactions	N. users	N. items	Sparsity	Aux. info.
MoviesLens-1M	1,000,209	6,040	3,706	95.53%	✓
MovieLens-100K	100,000	943	1,682	93.7%	✓
Pinterest	1,500,809	55,187	9,916	99.73%	✗
Book-Crossing	1,149,780	278,858	271,379	99.9%	✓
Advertising	880,596	448,158	737	99.7%	✓
Netflix	15,348,808	406,261	9,228	99.1%*	✓
Citeulike-a	19,107	5,551	16,980	99.9%	✗
Citeulike-t	52,946	7,947	25,975	99.78%	✗

Table 1. Statistics of datasets used in the reviewed papers

**The sparsity ratio on the Netflix dataset is of the original dataset, as no details were given on the combined dataset.*

3. Challenges of Collaborative Filtering

Recommendation systems that use collaborative filtering must deal with several challenges, which we will briefly discuss along with known solutions, if any exist.

Scalability – the number of existing users and items on some systems might grow very large. Thus, even algorithms with seemingly good complexity ($O(M)$, for example) might still require a lot of computation resources. Additionally, systems usually need to quickly react to changes or provide new predictions (a user looks at a new item, causing both an update to the rating matrix, while also requiring new recommendations).

Dimensionality reduction techniques can deal with this challenge as well as quickly provide new predictions but might undergo computationally expensive factorization on updates. However, some incremental factorization algorithms were suggested to combat the issue [20].
CF algorithms can also utilize clustering methods to address this challenge by greatly reducing the number of users or items that will be processed for recommendation. They might also only look at users with co-rated items in the first place.
Of course, all these solutions come with a tradeoff between the scalability they provide and their performance.

Sparsity – most real systems work with very large item sets. Thus, the rating matrix will be extremely sparse (see the sparsity ratios of the real-life datasets presented above). This is usually the cause for few issues:
The cold start problem (also dubbed new user/item problem) which occurs when a new user or item has entered the systems. New items cannot be recommended until some users rate them, and it will be difficult to find good, similar users to a new user with few ratings.
Reduced coverage is a related problem, as in a system of many items and the number of users' ratings is very small, the recommender system may be unable to recommend most of the items.
The neighbor transitivity problem, in which users with similar tastes may not be identified if they haven't rated any of the same items.

Dimensionality reduction techniques can greatly reduce the rating matrix's size by focusing on important features of users or items. Many model-based algorithms rely on some version of dimensionality reduction.
These techniques, however, might ignore important information for recommendation, especially if the target dimension is too small [6] [21].
Some Hybrid methods were found helpful as well to address this challenge. By using auxiliary information, even new users or items can receive good predictions, solving the cold start problem and offering larger coverage of items.

Synonymy – mostly challenging methods that process item or user information "as-is", synonymy is the tendency of a similar information to have completely different entries.

This challenge relates to any system that does any kind of natural language processing (NLP).
Recently, using advanced algorithms and the help of machine learning, methods begun to perform well enough for practical use [22], but are still far from perfect.

Shilling Attacks – in cases where anyone may provide ratings, parties with selfish interest may use this to alter recommendations to their favor, by giving fake positive reviews to their own items, or fake negative reviews to their competitors.

There has been some research and identification of these kind of attacks [23] [24], and some solutions were suggested, such as removing global effects in the data normalization stage in a memory-based CF method [25].

Smaller Challenges – some challenges are smaller in the sense of their prevalence in CF-based system.
Grey Sheep refers to user whose ratings to not agree or disagree with any other group of

users and thus do not benefit from CF. Some Hybrid approaches combining CF and information-based methods can deal with these users by adapting the weight based on identification of these users [26].
Working with real users' ratings might raise privacy concerts for CF algorithms. There has been work aimed to change CF algorithm to work with private data [27].
Lastly, explain-ability is an important aspect to recommendation system, giving users intuitive reasoning as to why specific recommendations were given [28]. Some CF methods might not be able to produce such information.

4. Example architecture of a recommendation system

In this section, we will describe an architecture of a recommendation system, based on the first representative method – the user-based neighborhood algorithms, discuss its complexity and the challenges it faces.

We imagine a system where any user can explicitly interact with any item, and sometimes need to be provided with their top K recommendations (for example, a website such as IMDB, which supports rating a movie and providing K similar movies based on those a user likes).

As with all recommendation systems, we require four functionalities, each for an important event – a new user or item enters the system, a user interacts with an item explicitly, and a user requires their top-K recommendations.

Our architecture will save an $N \times M$ interactions matrix, containing values for users and items that had interacted (a fixed values of some explicit rating, and "null" values to indicate no interaction).

New user/item – A new, "null" row/column will be added to the matrix, respectively.

Interaction between user u and item i – We will update the appropriate entry u, i in the matrix with the relevant value.

Top-K item recommendations for user u – as defined in the formulas in the relevant section, we first calculate the similarities between u and all other users ($w_{u,v}$). Then, we iterate over the set of items that u had not interacted with, calculate the weighted sum to predict a value for each of them ($\tilde{r}_{u,i}$). During this process, we only save the top K scoring items, and finally provide them to u.

Complexity-wise, the first three functions can be done in $O(1)$, but recommendations require calculating similarities and predictions, for which we need to iterate over all users and, in the worst-case where u had interacted with small number of items – we need a total of $O(N \cdot M)$ time.
It is worth mentioning that we can freely move the similarity calculations and item predictions between the recommendation and interaction functions, which might improve complexity, depending on the functionality that happen more frequently.
Saving the interaction matrix will require $O(N \cdot M)$ space, and under this architecture, there is no way around it. With a small change – to generate this matrix only on predictions – we can save only a sparse representation of it.

As can be seen from the complexity discussion above, our architecture will not handle large number of items or users within reasonable amount of time, and thus is not scalable and not robust to a sparse rating matrix (and specifically, susceptible to the cold start problem). It is also vulnerable to shilling attacks, overall emphasizing the challenges memory-based CF algorithms face.

5. Experimental evaluation

In this section, we will evaluate two CF methods.
We are mostly interested to analyze the strengths or weaknesses of the memory-based and model-based methods, as well as examine the premise of deep networks.

Thus, we kill two birds with one stone, by picking the two representative methods of memory-based and model-based types described above and analyze their performance.

I. Method 1 – User-based Pearson Correlation Weighted Average Sum (UPWA)

This method was implemented, and the results presented in a paper by *Krishnan et al.* [29] discussing neighborhood-based methods for the **MovieLens-100K** dataset were reproduced (figure 8).

The metric used to test the predictions on the dataset is Mean Absolute Error (MAE):

$$MAE = \frac{1}{|T|} \sum_{(u,i,r_{u,i}) \in T} |\tilde{r}_{u,i} - r_{u,i}|$$

Where T is the group of triplets indicating test cases of user, item, and true rating, that the implementation had not seen while training.

The full implementation can be viewed in the file `upwa.py`[4]. Few details about it:
The training function simply saves the rating matrix, and then the predict function does the heavy process of calculating similarities between all relevant users and u, and the weighted sum of users' ratings.

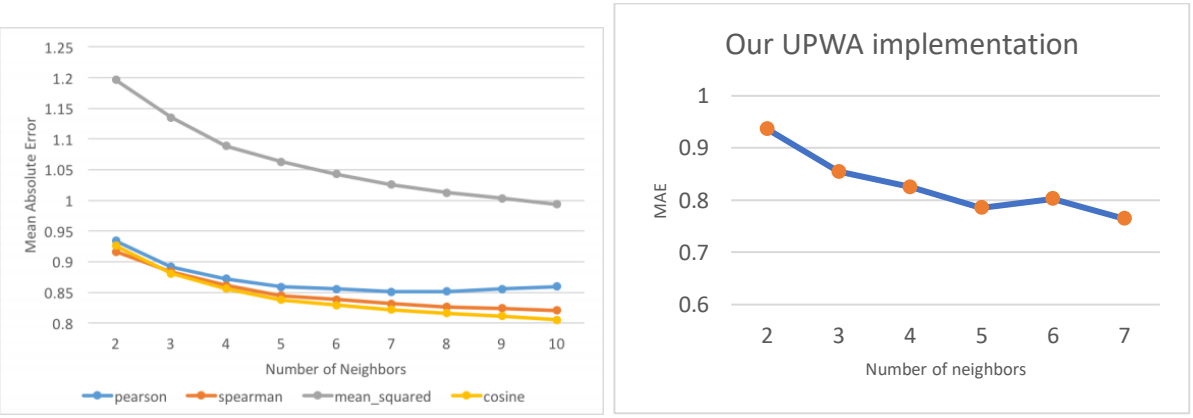


Figure 7. MAE versus number of neighbors. On the left, as taken from the paper, with the blue line indicating Pearson correlation, and on the right, our implementation, showing similar results.

II. Method 2 – Neural CF, Multi-Layer Perceptron (NCF-MLP)

This method was implemented, and the results presented in a paper by *He et al.* [1] for the **MovieLens-1M** dataset were reproduced (figure 9).

The metric used to test the predictions on the dataset is Hit Ratio at 10 (HR@10):
For each test case $(u, i, r_{u,i})$, the metric is calculated by taking 99 random, negative samples from the dataset (i.e.: items for which the user had not rated), and i, combining then into a group I, then predicting $\tilde{r}_{u,j}$ for all $j \in I$, and using the predicted rating to sort I, from best to worst. If the item i is within the top 10 rated items, we have a hit, and otherwise a miss.
The hit ratio is the number of hits divided by the number of test cases.

[4] This and any additional files can be found in the project's GitHub page.

The full implementation can be viewed in the file `ncf_mlp.py`[5]. Few details about it:
The deep-learning model is built exactly like presented in the paper. The train function sets aside some of the data it receives as an evaluation (validation) set, and runs for given amount of epochs, each has one round of fitting to the training data, then evaluating the model on the validation set to get the HR@10 metric, exactly like explained in the paper.
We also saved part of the dataset before training to be used as test set for additional metrics, and generated the required negative samples beforehand.
Due to slightly limited computational powers, we only trained on 500K out of the 1M samples, which we attribute the difference in HR@10/loss results to.
Additionally, we implemented an early stop methodology, whereas if we haven't improved the hit ratio in certain amount of epochs, we stop the training early.
Finally, the factors mentioned in the figure taken from the paper is the size of the last layer in the MLP, that was built using 3 layers, halving their sizes sequentially. Hence, a network with 8 factors will have the layers $32 \rightarrow 16 \rightarrow 8$.

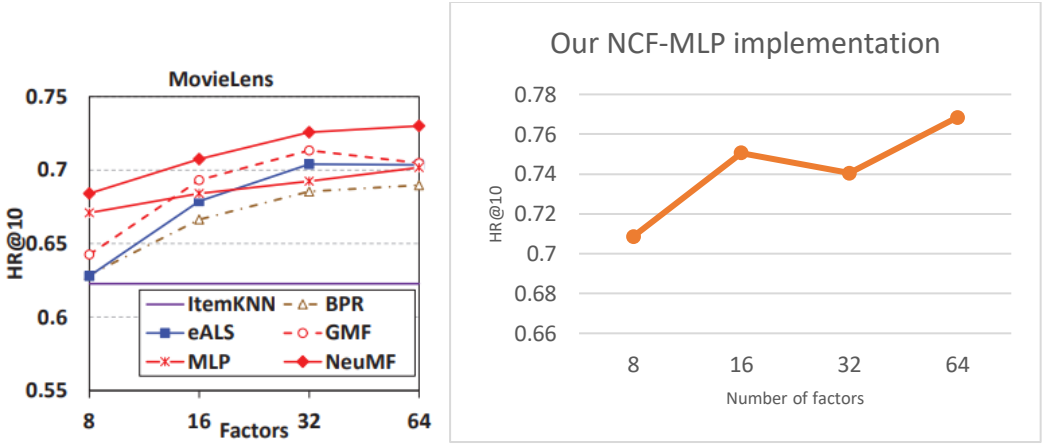


Figure 8. HR@10 versus the number of factors. On the left, as taken from the paper, with the red line with x's indicating MLP, and on the right our implementation, showing similar results.

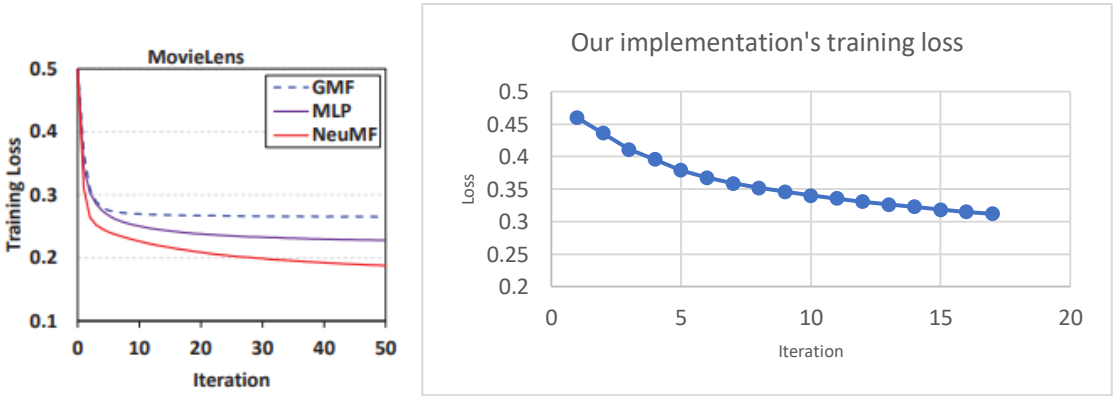


Figure 9. Training loss versus iteration (epoch), on the left as taken from the paper, with the purple line indicating MLP, and on the right the result from out implementation.

III. Comparison

Here we use an additional metric for comparison, Root Mean Squared Error (RMSE):

$$RMSE = \sqrt{\frac{1}{|T|} \sum_{(u,i,r_{u,i}) \in T} \left(\tilde{r}_{u,i} - r_{u,i}\right)^2}$$

[5] This and any additional files can be found in the project's GitHub page.

And the tests group, T is defined as before.

Since both papers originally explore these methods use the same dataset for evaluation, we only do two dataset comparisons. The first is the joint dataset, **MovieLens-100K**, using the same metric, and the second is **Jester-100K**[6], a dataset taken from a different domain – rating of jokes by users – from -10 to 10, having 7699 users and 159 jokes, with about 100,000 samples.

We note that due to the usage of the Sigmoid function as the activation of the final layer in the NCF-MLP model, we had to normalize ratings of the datasets used. We used min-max scaling to do so, using the minimum and maximum values of ratings that exists in the dataset. (In **MovieLens-100K**, we divided by 5, and in **Jester-100K** we added 10 and divided by 20).

MovieLens-100K – MAE and RMSE

As seen from table 2, NCF-MLP has slightly better results, but is not decisively better. Despite this, when we first tried to compare the methods on the **MovieLens-1M** dataset instead, UPWA was not even able to process it, expecting to finish in over 40 hours of work, pointing at the scalability problem of memory-based algorithms (table 3).

Method\Metric	MAE	RMSE
UPWA (5 neighbors)	0.16132	0.20569
NCF-MLP (8 factors)	0.14737	0.18811

Table 2. Comparison of both methods on the MovieLens-100K dataset

Method\Metric	MAE	RMSE
UPWA (5 neighbors)	-	-
NCF-MLP (8 factors)	0.31310	0.39642

Table 3. NCF-MLP results on the MovieLens-1M dataset (8 factors)

Jester-100K – MAE and RMSE

Now, the strength of deep learning-based methods begins to show. As seen from table 4, the deep learning approach performs better than the neighborhood-based method with larger margin.

We attribute this to the sparsity of the dataset. Whereas **MovieLens-100K** is approximately 95% sparse, **Jester-100K** is "just" 89% sparse, allowing the deep-learning model to utilize the additional info to find more intricate relations between users, that the basic Pearson correlation does not accomplish.

Method\Metric	MAE	RMSE
UPWA	0.20572	0.26935
NCF-MLP	0.17097	0.23161

Table 4. Comparison of both methods on the Jester-100K dataset

Experiments conclusion

The experiments held confirm some of the problems the deep-learning CF approaches solves – they achieve similar results (and sometimes better) compared to the classical, neighborhood-based methods, but can easily handle large amount of data.

Furthermore, the deep-learning approach shows it can extract finer details about the relations between users and items (while also combining information from both, simultaneously), than classical methods.

---

[6] The Jester dataset 4 for recommender systems research, (April 2015 – Nov 2019).

6. References

[1] He, Xiangnan, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. "Neural collaborative filtering." In *Proceedings of the 26th international conference on world wide web*, pp. 173-182. 2017.

[2] Li, Sheng, Jaya Kawale, and Yun Fu. "Deep collaborative filtering via marginalized denoising auto-encoder." In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pp. 811-820. 2015.

[3] Wang, Hao, Naiyan Wang, and Dit-Yan Yeung. "Collaborative deep learning for recommender systems." In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1235-1244. 2015.

[4] Su, Xiaoyuan, and Taghi M. Khoshgoftaar. "A survey of collaborative filtering techniques." *Advances in artificial intelligence* 2009 (2009).

[5] Ricci, Francesco, Lior Rokach, and Bracha Shapira. "Introduction to recommender systems handbook." In *Recommender systems handbook*, pp. 1-35. Springer, Boston, MA, 2011.

[6] Linden, Greg, Brent Smith, and Jeremy York. "Amazon. com recommendations: Item-to-item collaborative filtering." *IEEE Internet computing* 7, no. 1 (2003): 76-80.

[7] Sarwar, Badrul, George Karypis, Joseph Konstan, and John Riedl. "Item-based collaborative filtering recommendation algorithms." In *Proceedings of the 10th international conference on World Wide Web*, pp. 285-295. 2001.

[8] Resnick, Paul, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. "GroupLens: an open architecture for collaborative filtering of netnews." In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pp. 175-186. 1994.

[9] Do, Minh-Phung Thi, D. V. Nguyen, and Loc Nguyen. "Model-based approach for collaborative filtering." In *6th International Conference on Information Technology for Education*. 2010.

[10] Aggarwal, Charu C. "Model-based collaborative filtering." In *Recommender systems*, pp. 71-138. Springer, Cham, 2016.

[11] Takács, Gábor, István Pilászy, Bottyán Németh, and Domonkos Tikk. "Investigation of various matrix factorization methods for large recommender systems." In *2008 IEEE International Conference on Data Mining Workshops*, pp. 553-562. IEEE, 2008.

[12] Funk, Simon. "Netflix update: Try this at home." (2006).

[13] Koren, Yehuda, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems." *Computer* 42, no. 8 (2009): 30-37.

[14] Mnih, Andriy, and Russ R. Salakhutdinov. "Probabilistic matrix factorization." In *Advances in neural information processing systems*, pp. 1257-1264. 2008.

[15] Glorot, Xavier, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks." In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315-323. 2011.

[16] He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778. 2016.

[17] Kramer, Mark A. "Nonlinear principal component analysis using autoassociative neural networks." *AIChE journal* 37, no. 2 (1991): 233-243.

[18] Vincent, Pascal, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. "Extracting and composing robust features with denoising autoencoders." In *Proceedings of the 25th international conference on Machine learning*, pp. 1096-1103. 2008.

[19] Chen, Minmin, Zhixiang Xu, Kilian Weinberger, and Fei Sha. "Marginalized denoising autoencoders for domain adaptation." *arXiv preprint arXiv:1206.4683* (2012).

[20] Sarwar, Badrul, George Karypis, Joseph Konstan, and John Riedl. "Incremental singular value decomposition algorithms for highly scalable recommender systems." In *Fifth international conference on computer and information science*, vol. 1, no. 012002, pp. 27-8. 2002.

[21] Sarwar, Badrul, George Karypis, Joseph Konstan, and John Riedl. "Analysis of recommendation algorithms for e-commerce." In *Proceedings of the 2nd ACM conference on Electronic commerce*, pp. 158-167. 2000.

[22] Curran, James R., and Marc Moens. "Improvements in automatic thesaurus extraction." In *Proceedings of the ACL-02 workshop on Unsupervised lexical acquisition*, pp. 59-66. 2002.

[23] Lam, Shyong K., and John Riedl. "Shilling recommender systems for fun and profit." In *Proceedings of the 13th international conference on World Wide Web*, pp. 393-402. 2004.

[24] O'Mahony, Michael, Neil Hurley, Nicholas Kushmerick, and Guénolé Silvestre. "Collaborative recommendation: A robustness analysis." *ACM Transactions on Internet Technology (TOIT)* 4, no. 4 (2004): 344-377.

[25] Bell, Robert M., and Yehuda Koren. "Improved neighborhood-based collaborative filtering." In *KDD cup and workshop at the 13th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 7-14. sn, 2007.

[26] Miranda, Tim, Mark Claypool, Anuja Gokhale, Tim Mir, Pavel Murnikov, Dmitry Netes, and Matthew Sartin. "Combining content-based and collaborative filters in an online newspaper." In *In Proceedings of ACM SIGIR Workshop on Recommender Systems*. 1999.

[27] Canny, John. "Collaborative filtering with privacy via factor analysis." In Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 238-245. 2002.

[28] Koren, Yehuda. "Tutorial on recent progress in collaborative filtering." In *Proceedings of the 2008 ACM conference on Recommender systems*, pp. 333-334. 2008.

[29] Nirmal Krishnan, Emily Brahma, Survey of Neighborhood-based Collaborative Filtering Techniques for a Movie Recommendation Engine, 4 October 2016