Aalto University
School of Science

Collaborative Filtering in Recommendation Systems

Gadidjah Ögmundsdóttir

Muhammad Kamal Memon

July 2018

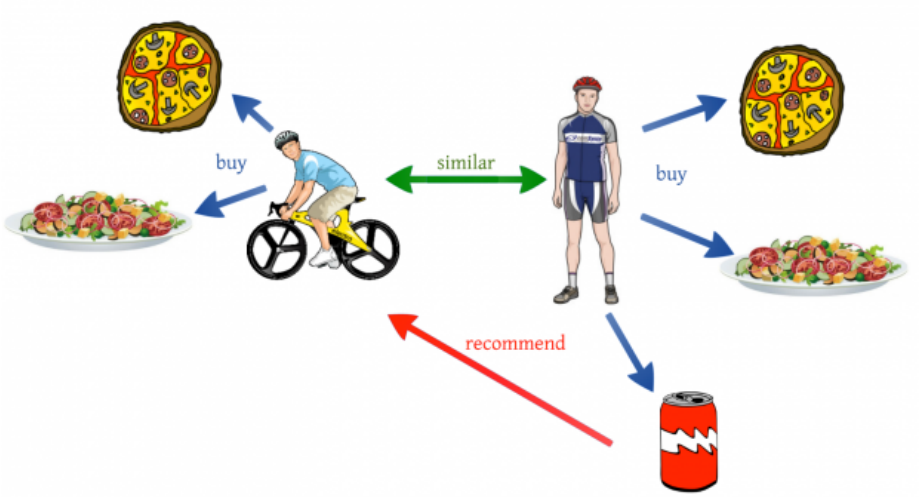
Individual Studies in Computer Science
CS-E4004
Aristides Gionis

Contents

1 Introduction

Recommendation systems are widespread in this age, from *Amazon* to *Netflix* and from *Youtube* to all major e-commerce sites, these systems are a vital part of their digital experience. Recommendation systems personalize users interactions by predicting what the user might be interested in. These predictions are based on user behavior patterns and the items of interest. One of the most successful approaches for building a recommendation engine is Collaborative Filtering. This approach is based on the simple idea that users like things that are liked by other users with a similar taste. Figure 1 shows this simple idea.

Figure 1: Collaborative recommendation example (Ved, 2016)



A wide variety of algorithms and mechanisms exists for implementing Collaborative Filtering. In this report we will discuss and explore some of these methods along with a small scale implementation on three different datasets.

2 Literature Review

Recommendation systems attempt to predict a user's response to an item by discovering similar items and the response of the user to those discovered items. Recommendation systems are separated into the two classes; content-based systems and systems which use collaborative filtering. Content-based systems measure similarity by looking for common features of items while collaborative filtering systems measure the similarity of users by their item preferences and/or measure similarity of items by the users who like them (Leskovec, Rajaraman, & Ullman, 2014).

These recommendation systems make use of utility matrices, item profiles and user profiles. The utility matrices are used to store known information about the degree to which a user likes an item. Most often the entries are unknown as the system has only received explicit feedback for a limited amount of items per user, so an essential problem of recommending items to users is to predict values of the unknown entries of the utility matrix based on the values of the known entries. Item profiles would consist of features of items on which content-based similarity can be based. User profiles can be constructed by a content-based collaborative filtering system by measuring the frequency with which features appear in the items the user likes. Then it is possible to estimate the degree to which a user will like an item by the similarity of the item's profile to the user's profile (Leskovec et al., 2014).

There exists multiple algorithms and approaches for doing collaborative filtering and the most common method for comparing the learning rates of different algorithms is to compare the quality versus the number of ratings. Also a major aspect of this is the sparsity of the data, as in data-sparse domains the recommender algorithms learning rates become a much more significant evaluation factor. Moreover its not just the implementation of the system but also its effective outcome that should be considered a metric when evaluating such a system. This output has two conflicting dimensions when viewed from a user's perspective. First that what is the strength of the recommendation: how much does the user like the recommended item and secondly how much the system itself is sure that the recommendation is accurate (Herlocker, Konstan, Terveen, & Riedl, 2004).

The data itself is of course the most crucial part in comparing these systems, properties like whether the ratings are implicit or explicit, their scale and dimensions impact the quality of the system directly. The accuracy metrics which can be quantified in comparative sense can be predictive metrics like mean absolute errors and/or classification metrics which measure the frequency with which a recommender system makes correct or incorrect decisions about whether an item is good or not. Other metrics include; precision and recall, ROC curves, rank accuracy and ad hoc classification accuracy metrics (Herlocker et al., 2004).

However, some recommender systems produce highly accurate recommendations—and yet are useless for practical purposes. For example, a recommendation for a grocery store user might have bananas as the top item. Statistically, this recommendation is highly accurate: almost everyone buys bananas however, everyone who comes to a grocery store to shop has bought bananas in the past, and knows whether or not they want to purchase more anyway. Hence there is a need for analyzing recommender systems

that consider the "nonobviousness" of the recommendation. One such dimension is *novelty*, which has been addressed in information retrieval literature. Another related dimension to this value is called serendipity. A recommendation with high serendipity will give a user a surprisingly interesting item he might not have otherwise discovered (Baeza-Yates & Ribeiro-Neto, 1999).

There has been research about novelty and serendipity in the context of collaborative filtering systems. A simple modification which is suggested is to create a list of obvious recommendations, and remove them from output before presenting it to users. On the other hand its very difficult to design metrics to measure serendipity as it is a measure of the degree to which the recommendations are both attractive and surprising to the users (Sarwar, Karypis, Konstan, & Riedl, 2001).

3 Methods of Collaborative Filtering

Methods of collaborative filtering needs to take into account that most of the ratings are unobserved or missing. The unspecified ratings are assigned in various ways with differing collaborative filtering methods. Most collaborative filtering models center on utilizing inter-item correlations or inter-user correlation when predicting unspecified ratings. Some models use both correlations and others use carefully designed optimization techniques to create a training model where the model is then used to assign missing values in the matrix. Commonly used methods in collaborative filtering are referred to as memory-based methods and model-based methods. Figure 2 shows the advantages and limitations of each of the methods (Aggarwal, 2016).

Figure 2: Overview of CF techniques (kumar Bokde, Girase, & Mukhopadhyay, 2015)

Method	Advantages	Limitations
Memory-based CF	- Easily implemented - New data is easily and incrementally added - Content of recommended items does not need consideration - Good scalability with correlated items	- Dependent on human ratings - New user and new item have a cold start problem - Rating matrix has sparsity problem - Large datasets have limited scalability
Model-based CF	- Addresses the sparsity and scalability problems better - Prediction performance is improved	- Expensive model building - Trade-off between prediction performance and scalability - Information loss in dimensionality reduction techniques

3.1 Memory-based methods

Memory-based methods are one of the oldest collaborative filtering algorithms, in which the ratings of user-item combinations are predicted based on their neighborhoods. The neighborhoods can be defined in user-based or item-based ways. Memory-based techniques have the advantage that they are simple to implement but they do not work too well with sparse ratings matrices.

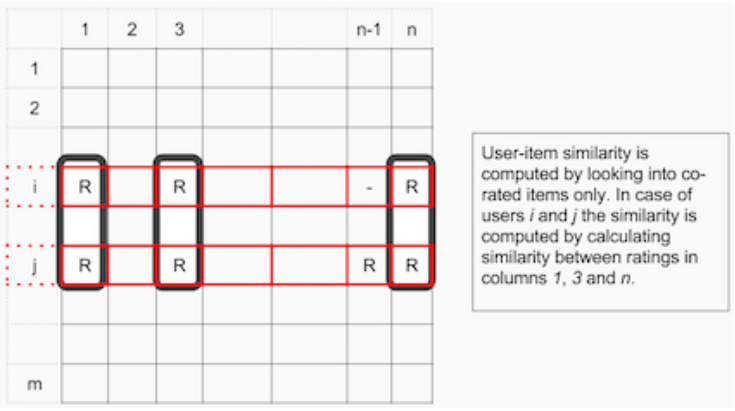
3.1.1 User-based collaborative filtering

In user-based collaborative filtering, ratings provided by users with similar preferences of a specific target user are used to make recommendations for the target user. The goal is to determine users, who are similar to the the target user, and recommend ratings for the unobserved ratings of the target user. The k most similar users to the target

user can be used to make the rating predictions. Discovering similar users is done with similarity functions computed between the rows of the ratings matrix.

The similarity values between users are measured by observing all the items that are rated by both users as can be seen on Figure 3.

Figure 3: User based collaborative filtering (Johannsdottir, n.d.)

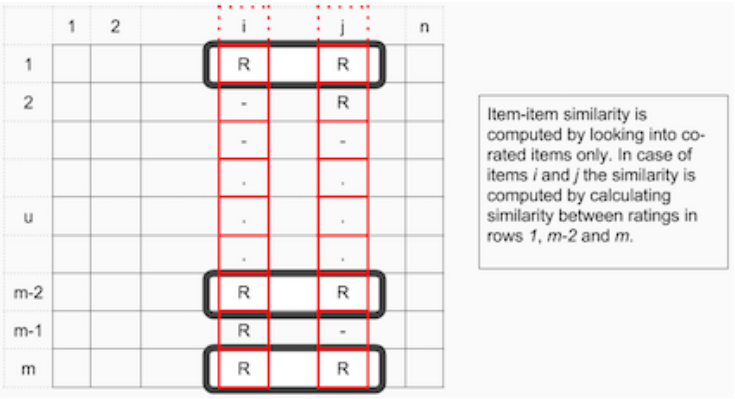


3.1.2 Item-based collaborative filtering

In item-based collaborative filtering a set is determined which contains items that are most similar to the target item. The ratings in the set, which were specified by the user, are used to predict whether the user will also like the target item. To discover similar items, similarity functions are computed between the columns of the ratings matrix.

The similarity values are measured by observing all the users who have rated both items as can be seen on Figure 4.

Figure 4: Item based collaborative filtering (Johannsdottir, n.d.)



3.2 Model-based methods

Model-based methods utilize machine learning and data mining methods for predictive models. Model-based methods are better than memory-based methods in regards to space-efficiency, training speed and prediction speed and avoiding over-fitting (Aggarwal, 2016).

Model-based methods consist of creating a model that generates the recommendations and aim to uncover latent factors that explain observed ratings. Examples of model-based collaborative filtering algorithms include Bayesian Networks, Clustering Models and Latent Semantic Models. A difference between memory-based and model-based collaborative filtering methods is that the latter does not use the whole dataset to compute predictions for real data (kumar Bokde, Girase, & Mukhopadhyay, 2015).

Following are brief descriptions of the methods we have implemented:.

3.2.1 Matrix Factorization

Matrix Factorization (MF) is a powerful approach which enables us to see the hidden structure behind the data. MF models are based on the latent factor model. In this approach the rating matrix is modelled as the product of user factor matrix and an item factor matrix. In its basic form MF characterizes user and items by vectors of factors derived from the rating patterns. It maps both user and items to a joint latent space where the interaction (ratings) are the inner products of that space. It has been found as one the best approaches to reduce the problem of high level of sparsity of the data.

MF models create a latent factor space with dimensionality d to map both users and items. In this joint space each user-item interaction is modeled as an inner product. Accordingly, for a given item $i \in \mathcal{I}$ and user $u \in \mathcal{U}$, vectors $q_i \in \mathbb{R}^d$ and $p_u \in \mathbb{R}^d$ are used to represent item i and user u respectively. The vector q_i measures how positively or negatively item i exhibits the latent factors that are used to represent the item space. Similarly, the vector p_u measures the extent to which user u prefers each factor mentioned in the corresponding q_i (Koren, Bell, & Volinsky, 2009)

Following are the models that we have implemented:

- **Singular Value Decomposition (SVD):** SVD is basically a powerful approach for dimensionality reduction. The key problem SVD solves is that it finds a lower dimensional feature space of matrix. It tends to transform the ratings matrix and once it is completed, the users and items can be considered as points in the new k-dimensional space.

  Matrix factorization models map both users and items to a joint latent factor space of dimensionality f, such that user-item interactions are modeled as inner products in that space. Accordingly, each item i is associated with a vector $q_i \in \mathbb{R}^f$, and each user u is associated with a vector $p_u \in \mathcal{R}^f$. For a given item i, the elements of q_i measure the extent to which the item possesses those factors, positive or negative. For a given user u, the elements of p_u measure the extent of interest the user has in items that are high on the corresponding factors, again, positive or negative. The resulting dot product, $q_i^{\mathcal{T}} p_u$, captures the interaction between user u and item i—the user's overall interest in the item's characteristics.

This approximates user u's rating of item i, which is denoted by r_{ui}, leading to the estimate

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

The major challenge is computing the mapping of each item and user to factor vectors q_i, $p_u \in \mathbb{R}^f$. After the recommender system completes this mapping, it can easily estimate the rating a user will give to any item by using Equation 1. To learn the factor vectors (p_u and q_i), the system minimizes the regularized squared error on the set of known ratings:

$$\min_{q*,p*} \sum_{(u,i)\in k} (r_{ui} - q_i^{\mathcal{T}} p_u)^2 + \lambda(||q_i||^2 + ||p_u||^2) \tag{2}$$

Here, k is the set of the (u, i) pairs for which r_{ui} is known (the training set).

The system learns the model by fitting the previously observed ratings. However the goal is to generalize those previous ratings in a way that predicts future, unknown ratings. Thus, the system should avoid overfitting the observed data by regularizing the learned parameters, whose magnitudes are penalized. The constant λ controls the extent of regularization and is usually determined by cross validation. The minimization in Equation 2 is performed by a very straightforward stochastic gradient descent:

$$\begin{aligned} b_u &\leftarrow b_u + \gamma(e_{ui} - \lambda b_u) \\ b_i &\leftarrow b_i + \gamma(e_{ui} - \lambda b_i) \\ p_u &\leftarrow p_u + \gamma(e_{ui} \cdot q_i - \lambda p_u) \\ q_i &\leftarrow q_i + \gamma(e_{ui} \cdot p_u - \lambda q_i) \end{aligned}$$

where $e_{ui} = r_{ui} - \hat{r}_{ui}$. These steps are performed over all the ratings of the trainset and repeated *n_epochs* times. User and item factors are randomly initialized according to a normal distribution. (Koren et al., 2009)

- **Non-Negative Matrix Factorization (NMF):** NMF is a group of algorithms in multivariate analysis where a matrix X is factored into two matrices with the property that all three matrices have no negative elements. As compared to other approaches NMF take advantage of the fact that most real world data is non-negative like images or videos and maintain such constraints in factorization (kumar Bokde et al., 2015).

  This algorithm is very similar to SVD. The prediction $\hat{r}_{ui}$ is set as:

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

  where user and item factors are kept positive. The optimization procedure is a (regularized) stochastic gradient descent with a specific choice of step size that ensures non-negativity of factors, provided that their initial values are also positive.

At each step of the SGD procedure, the factors f or user u and item i are updated as follows:

$$p_{uf} \leftarrow p_{uf} \cdot \frac{\sum_{i \in I_u} q_{if} \cdot r_{ui}}{\sum_{i \in I_u} q_{if} \cdot \hat{r}_{ui} + \lambda_u |I_u| p_{uf}}$$
$$q_{if} \leftarrow q_{if} \cdot \frac{\sum_{u \in U_i} p_{uf} \cdot r_{ui}}{\sum_{u \in U_i} p_{uf} \cdot \hat{r}_{ui} + \lambda_i |U_i| q_{if}}$$

where λ_u and λ_i are requlariza tion parameters.

3.2.2 Clustering

The idea behind using clustering approaches in Collaborative Filtering is very intuitive. We want to group users into clusters based on the ratings or their like/dislike of the items they have interacted with. This allows to recommend new items to the same group of people. However finding optimal clusters is tricky because the group of items should be used to determine the users and vice versa (Ungar et al., 1998). In this report we will work with a variation of K-Nearest neighbour algorithm; **KNNWithMeans**. It is a basic collaborative filtering algorithm, taking into account the mean ratings of each user.

The prediction $\hat{r}_{ui}$ is set as:

$$\hat{r}_{ui} = \mu_u + \frac{\sum\limits_{v \in N_i^k(u)} \text{sim}(u, v) \cdot (r_{vi} - \mu_v)}{\sum\limits_{v \in N_i^k(u)} \text{sim}(u, v)}$$

Here the similarity function *(sim)* we are using is Mean Squared Difference (MSD) that is the similarity between all pairs of users. k is the number of neighbours to take into account for aggregation.

3.2.3 Deep Learning

There is not much research or material available online on the use of deep learning models for collaborative filtering. We get this idea by going through the different approaches of matrix factorization that we can employ neural nets in learning the hidden latent factors of a utility matrix.

The deep learning approach is similar or can be termed as an an extension to matrix factorization method. For example in SVD, we decompose our original sparse matrix into product of 2 low rank orthogonal matrices. For neural net implementation, we don't need them to be orthogonal, we want our model to learn the values of embedding matrix itself. (Grover, 2017)

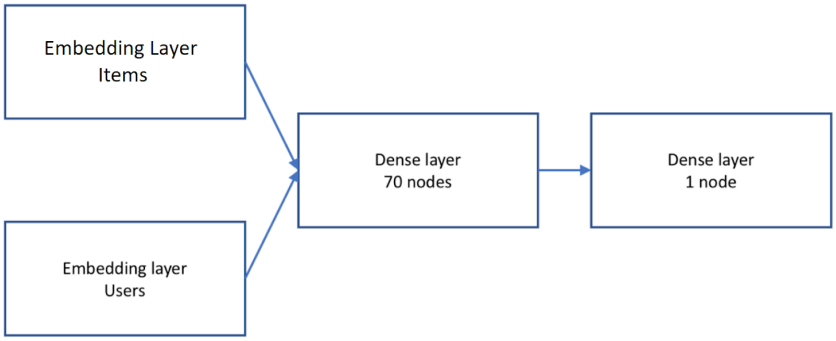
3.2.3.1 Architecture

For this report we have used *Fast.ai* deep learning library which provides ready-made neural nets for Collaborative Filtering. Fast.ai's neural network consist of 4 layers: two

embedding layers, one for the users and one for the items, and then two dense layers where the computations took place. These layers are shown in Figure 5.

These embedding layers are crucial to the neural network's performance, improving the RMSE score significantly. They allow a bias to be added to the neural network, which considers (on top of how users are rating item) differences across users. The dense or fully connected layers are fairly straightforward; "dense" means that every node in the layer is connected to every input layer. They perform a linear operation in which every input is connected to every output by a weight (so there are *n_inputs* x *n_outputs* weights) performing the matrix product. An embedding layer takes the inputs (so for the movies, this would be the rankings of the users) and turns it into a high-dimensional vector. It considers all users' ratings when creating this high dimensional vector, thus accounting for the "goodness" or "badness" of the item, as acknowledged by all users (Tseng, 2017).

Figure 5: Embedding layers of deep learning method



For instance, in terms of books, *The Great Gatsby* is universally acclaimed as a good book; whether or not someone enjoys fiction, they are more likely to enjoy this book. Similarly, in terms of users, if one user dislikes jokes in general, he is going to rate the jokes lower even if they line up with his preferences.

The user and item latent features are looked up from the embedding matrices for specific user-item combination. These become the input for next dense layer. The dense layer is followed by an activation function - *softmax*. We learn the corresponding weights by optimization. In our case we are using Adam as our optimization algorithm.

4 Datasets

We are implementing above mentioned methods of Collaborative Filtering on the following three datasets with empirical evaluation using Root Mean Square error as the metric:

- Book-Crossing
- MovieLens
- Jester

4.1 Book-Crossing

Book-Crossing dataset contains ratings of books from the readers. This dataset is collected by Cai-Nicolas Ziegler from the *Book-Crossing community*. It contains 105,283 unique users, anonymized but with demographic information providing 1,149,780 ratings to 340,240 books (Ziegler, McNee, Konstan, & Lausen, 2005).

The data set holds three tables; User, Books and Ratings. We are mostly interested in the ratings table. Samples from the ratings table are shown in Figure 6 and the fields of the book-crossing ratings table are shown in Table 1.

Figure 6: Book-Crossing ratings samples

	User-ID	ISBN	Book-Rating
183558	108405	202648	9
403889	255056	241252	10
44872	23902	170127	7
433443	276641	135336	7
395343	249862	155496	10

Table 1: Book-Crossing ratings table

Field	Type	Description
User-ID	big-int	Unique identifier of a reader
ISBN	string	Unique identifier of a book
Book-Rating	int	The rating reader gave to the book. Ranges from 0-10

For data pre-processing, we clean the ISBN column with non-unicode characters using regex. When we do more exploratory analysis we find that there are some records which have Book-Rating as 0. This zero (0) rating means the the reader hasn't rated the book. For doing collaborative filtering this can become an issue as it will act as decreasing the overall rating of the book. Hence we removed the books who doesn't have any rating by any user and for others we replaced the zero rating with the average rating of the book. Also for ISBN field we used a label encoder to encode the values in numeric format. At this point our data set is pre-processed and is ready to perform experiments on.

4.2 MovieLens

MovieLens dataset was collected by the GroupLens Research Project at the University of Minnesota. It has ratings of movies by users on a scale from 1 to 5 (Harper & Konstan, 2015).

We are using the MovieLens 100K stable benchmark dataset. It consists of:

- 100,000 ratings (1-5) from 943 users on 1682 movies with a unix timestamp.
- Each user has rated at least 20 movies.
- Simple demographic info for the users (age, gender, occupation, zip)

Samples from the ratings table are shown in Figure 7. Since its a cleaned dataset it has no N/A values and is ready for performing experiments on.

Figure 7: Movielens ratings samples

user_id	movie_id	rating	timestamp
452	223	5	885816768
383	135	5	891193042
256	974	3	882164059
279	709	4	875310195
117	164	5	881011727

4.3 Jester

Jester dataset is anonymous ratings data from the Jester Online Joke Recommender System. The users have rated jokes on real values ranging from -10.00 to +10.00.

The dataset consists of:

- 1,000,000 ratings of jokes on a scale of -10.00 to +10.00

- 63978 unique users
- 150 jokes

Samples from ratings table are shown in Figure 8.

Figure 8: Jester ratings samples (Goldberg, Roeder, Gupta, & Perkins, 2001)

user_id	joke_id	rating
47262	35	-8.031
33089	125	2.750
56729	17	9.844
6791	7	-8.312
12671	8	5.688
36061	120	-2.438

5 Implementation

5.1 Methodology

For matrix factorization, clustering and cross validation we are using ***Surprise*** package which is a Python scikit for building and analyzing recommender systems. And to perform deep learning we used ***fast.ai*** which is a developer friendly deep learning library in Python.

To compare different methods of Collaborative filtering we have implemented and evaluated key methods on a predictive accuracy metric which is Root Mean Square error. However, we are only interested in implementing the model based methods as they serve well in uncovering the latent factors present in the data. Also they are more challenging to implement and need fine tuning over hyperparameters to give out the best result.

We concern ourselves with the ratings table of datasets as they have the user/item rating information on which we base the Collaborative Filtering models. The approach then is to perform grid search to find the best hyperparameters for each algorithm. After splitting data into three folds for cross validation the algorithms are run with the best parameters from grid search on the data. We are taking the Root Mean Squared Error as an evaluation metric for these methods. Figures 9-12 below show the hyperparameters of the respective methods.

Figure 9: Hyperparameters of SVD method

Hyperparameter	Description	Jester	MovieLens	Book-Crossing
n_factors	The number of factors	100	50	100
n_epochs	The number of iteration of the SGD procedure	20	15	30
init_mean	The mean of the normal distribution for factor vectors initialization	0	0	0
init_std_dev	The standard deviation of the normal distribution for factor vectors initialization	0.1	0.1	0.1
lr_all	The learning rate for all parameters	0.001	0.002	0.005
reg_all	The regularization term for all parameters	0.02	0.02	0.03

5.1.1 Recommendations

Once the model is trained we use that trained model to get the top-N recommendations for each user. For instance to get the top-10 items with highest rating prediction for each user in a dataset we predict all the ratings for the pairs (user, item) that are not in the training set. We then retrieve the top-10 prediction for each user. To do so, we

Figure 10: Hyperparameters of NMF method

Hyperparameter	Description	Jester	MovieLens	Book-Crossing
n_factors	The number of factors.	40	15	30
n_epochs	The number of iteration of the SGD procedure.	50	50	50
reg_pu	The regularization term for users λu.	0.06	0.02	0.05
reg_qi	The regularization term for items λi.	0.06	0.02	0.05
init_low	Lower bound for random initialization of factors.	0	0	0
init_high	Higher bound for random initialization of factors	1	1	1

Figure 11: Hyperparameters of KNN method

Hyperparameter	Description	Jester	MovieLens	Book-Crossing
k	the (max) number of neighbors to take into account for aggregation	40	20	40
sim_func	The name of the similarity function to use.	MSD	MSD	MSD
user_based	Whether similarities will be computed between users or between items.	TRUE	TRUE	TRUE

first map the predictions to each user and then sort the predictions and retrieve the 10 highest ones.

For example, in the MovieLens data set, using the SVD trained model we generate top-10 recommendations for a random user ($user\_id = 2000$), this is displayed in Figure 13.

As can be seen by the genre column, the recommendation list is dominated by drama movies. This suggest that users similar to this user has rated drama movies highly.

Figure 12: Hyperparameters of Deep Learning method

Hyperparameter	Jester	MovieLens	Book-Crossing
n_factors	20	50	50
embeddings	4	4	4
weight decay	0.0001	0.0001	0.0002
validation set	20	20%	20%
batch size	32	32	64
epochs	5	5	3
learning rate	0.001	0.001	0.01

Figure 13: Top-10 movie recommendations for a user

	movie_id	prediction	title	genres
0	953	4.868923	It's a Wonderful Life (1946)	Drama
1	668	4.866858	Pather Panchali (1955)	Drama
2	1423	4.859523	Hearts and Minds (1996)	Drama
3	3307	4.834415	City Lights (1931)	Comedy|Drama|Romance
4	649	4.802675	Cold Fever (Á köldum klaka) (1994)	Comedy|Drama
5	669	4.797451	Aparajito (1956)	Drama
6	326	4.784828	To Live (Huozhe) (1994)	Drama
7	3092	4.761148	Chushingura (1962)	Drama
8	3022	4.753003	General, The (1927)	Comedy
9	2351	4.720692	Nights of Cabiria (Le Notti di Cabiria) (1957)	Drama
10	926	4.719633	All About Eve (1950)	Drama

5.2 Results

5.2.1 Book-Crossing Dataset

Method and metric		Fold 1	Fold 2	Fold 3	Mean	Standard deviation
SVD	RMSE	1.6479	1.6461	1.6516	**1.6485**	0.0023
	Fit time	25.60	26.00	26.22	25.94	0.26
	Test time	1.95	1.61	1.60	1.72	0.17
NMF	RMSE	2.4404	2.4532	2.4369	**2.4435**	0.0070
	Fit time	44.04	46.31	44.63	44.99	0.96
	Test time	1.66	1.51	1.43	1.54	0.10
KNNWithMeans	RMSE	1.8602	1.8512	1.8627	**1.9780**	0.0049
	Fit time	59.46	72.43	74.88	68.92	6.76
	Test time	2.18	3.08	2.97	2.74	0.40

Deep Learning: Mean squared error on predictions = 1.8569

5.2.2 MovieLens Dataset

Method and metric		Fold 1	Fold 2	Fold 3	Mean	Standard deviation
SVD	RMSE	0.9444	0.9477	0.9435	**0.9452**	0.0018
	Fit time	5.07	5.04	5.47	5.19	0.20
	Test time	0.27	0.29	0.29	0.28	0.01
NMF	RMSE	0.9737	0.9702	0.9760	**0.9733**	0.0024
	Fit time	5.47	7.01	5.20	5.89	0.80
	Test time	0.41	0.29	0.27	0.32	0.07
KNNWithMeans	RMSE	0.9536	0.9609	0.9526	**0.9557**	0.0037
	Fit time	0.36	0.35	0.34	0.35	0.01
	Test time	5.06	5.11	4.96	5.04	0.06

Deep Learning: Mean squared error on predictions = 0.87488

5.2.3 Jester Dataset

Method and metric		Fold 1	Fold 2	Fold 3	Mean	Standard deviation
SVD	RMSE	4.3871	4.3857	4.3769	**4.3833**	0.0045
	Fit time	54.04	54.04	54.60	54.23	0.27
	Test time	5.86	4.10	4.29	4.75	0.79
NMF	RMSE	7.4298	7.4325	7.3367	**7.3997**	0.0445
	Fit time	7.29	7.66	7.58	7.51	0.16
	Test time	0.26	0.37	0.41	0.34	0.06
KNNWithMeans	RMSE	5.1290	5.1470	5.1131	**5.1297**	0.0138
	Fit time	102.37	104.42	108.00	104.93	2.33
	Test time	21.86	23.21	22.65	22.57	0.55

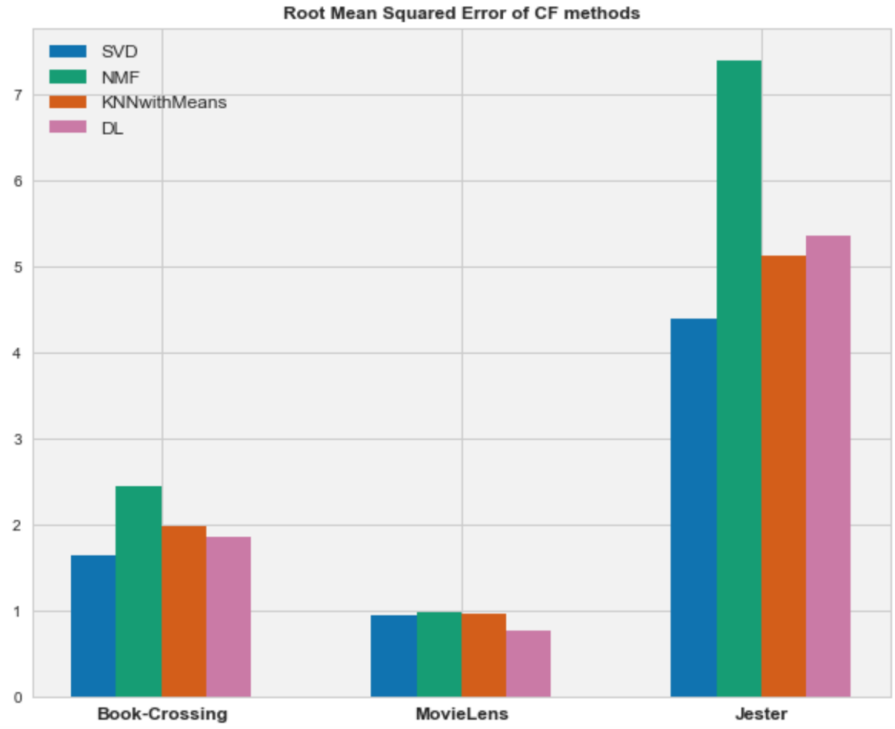
Deep Learning: Mean squared error on predictions = 5.358969

5.3 Comparison

Based on the results of our experimentation we can assert that the Singular Value Decomposition (SVD) and Deep Learning approach performed better than other methods. All four methods have performed well relatively on MovieLens dataset with DL approach and SVD giving out the least error score. For Book-Crossing dataset SVD works

the best and KNN and DL approach have somewhat similar results. For Jester dataset the relative difference between methods scores is high with SVD performing well as on other datasets. This comparison however is on a single predictive accuracy metric and doesn't fully reflects the different nuances of the recommender system as a whole but still however provide an empirical baseline for evaluation of methods. This comparison is shown in a bar graph in Figure 14.

Figure 14: Comparison of MSE scores using different CF methods on the three datasets



5.4 Code

The code for the implementation can be found at: https://github.com/kamalmemon/CollaborativeFiltering

6 Conclusions

Recommendation systems are abundant in our age and its immensely useful for user's digital experience. Most powerful and well researched method of building such recommendation system is Collaborative Filtering which is directly user targeted and bear fruitful results. In this report we have attempted to cover the basic types of Collaborative Filtering in which we learned that model based approach is well suited to data sparse domains and is also fine tuneable to the needs. We implemented the said methods on a small scale to draw comparisons between them and assess the results.

In our results' comparison Singular Value Decomposition (SVD) and Deep Learning approach comes out as the better approaches than other methods. This comparison however is on a single predictive accuracy metric and doesn't fully reflects the different nuances of the recommender system as a whole but still however provide an empirical baseline for evaluation of methods. Moreover Matrix Factorization comes across as the most viable way to find the hidden factors in a utility matrix built for collaborative filtering.

7 Future Work

To compare and evaluate different methods of collaborative there can be various deciding factors which can play a vital role in such a setting. The user tasks being evaluated, the types of datasets being used, the ways in which prediction quality is measured, the evaluation of prediction attributes other than quality, and the user-based evaluation of the system as a whole. On a large scale this task can be taken up as a future work in the context of comparing recommendation system.

Also the deep learning approach can be hugely customized for a specific dataset. We can also experiments with different architectures as there is a significant potential in using DL methods to improve the overall accuracy of the predictions. But such a task demands careful consideration, computation power and time and hence can be considered as a future work.

Last but not least we think that the problem of the consistency of algorithms and methods across different domains is also of prime interest. An algorithm may perform well on a particular dataset but might not be as effective in a different data setting. Hence comparing algorithms across a variety of different domains to understand the extent to which different domains are better served by different classes of algorithms is matter of interest which can also be taken as a future work from this report.

References

Aggarwal, C. C. (2016). *Recommender systems: The textbook*. Yorktown Heights, NY: Springer International Publishin Switzerland.

Baeza-Yates, R. A. & Ribeiro-Neto, B. (1999). *Modern information retrieval*. Boston, MA: Addison-Wesley Longman Publishing Co., Inc.

Goldberg, K., Roeder, T., Gupta, D., & Perkins, C. (2001). Eigentaste: A constant time collaborative filtering algorithm. *Inf. Retr. 4*(2), 133–151. doi:10.1023/A:1011419012209

Grover, P. (2017). Various implementations of collaborative filtering. Retrieved from https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0

Harper, F. M. & Konstan, J. A. (2015). The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst. 5*(4), 19:1–19:19. doi:10.1145/2827872

Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (2004). *Evaluating collaborative filtering recommender systems*.

Johannsdottir, A. (n.d.). Implementing your own recommender systems in python. Retrieved from https://cambridgespark.com/content/tutorials/implementing-your-own-recommender-systems-in-Python

Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, *42*(8), 30–37. doi:10.1109/MC.2009.263

kumar Bokde, D., Girase, S., & Mukhopadhyay, D. (2015). Role of matrix factorization model in collaborative filtering algorithm: A survey. *CoRR*, *abs/1503.07475*. arXiv: 1503.07475. Retrieved from http://arxiv.org/abs/1503.07475

Leskovec, J., Rajaraman, A., & Ullman, J. D. (2014). *Mining of massive datasets*. Palo Alto, CA.

Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on world wide web* (pp. 285–295). WWW '01. Hong Kong, Hong Kong: ACM. doi:10.1145/371920.372071

Tseng, G. (2017). Clustering and collaborative filtering — implementing neural networks. https://medium.com/@gabrieltseng/clustering-and-collaborative-filtering-implementing-neural-networks-bccf2f9ff988. Blog.

Ungar, L., Foster, D., Andre, E., Wars, S., Wars, F. S., Wars, D. S., & Whispers, J. H. (1998). Clustering methods for collaborative filtering. AAAI Press.

Ved. (2016). Recommender systems 101. Retrieved from https://d4datascience.wordpress.com/2016/07/22/recommender-systems-101/

Ziegler, C.-N., McNee, S. M., Konstan, J. A., & Lausen, G. (2005). Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on world wide web* (pp. 22–32). WWW '05. Chiba, Japan: ACM. doi:10.1145/1060745.1060754