Compressed knowledge transfer via Factorization models into Recommender Systems

A BACHELOR'S THESIS

Submitted in partial fulfillment
of the requirements for the award of degree

Of

BACHELOR'S OF TECHNOLOGY
In
INFORMATION TECHNOLOGY

Submitted by
Dhruv Kumar (IIT2012171)
Under the guidance of

Dr. Ratna Sanyal
Assistant Professor
IIIT-Allahabad

Prof. Artus Krohn-Grimberghe
Assistant Professor, AIS/BI
University of Paderborn

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY
ALLAHABAD – 211012 (INDIA)
March, 2016

Candidate's Declaration

I hereby declare that the work presented in this thesis entitled "Compressed knowledge transfer via factorization models into recommender systems", submitted in the partial fulfillment of the degree of Bachelor of Technology (B.Tech), in Information Technology at Indian Institute of Information Technology, Allahabad, is an authentic record of my original work carried out under the guidance of Prof. Artus Krohn-Grimberghe and Dr. Ratna Sanyal. Due acknowledgements have been made in the text of the thesis to all other material used. This thesis work was done in full compliance with the requirements and constraints of the prescribed curriculum.

Place: Allahabad

Date: 23/07/2016

_____

Dhruv Kumar

IIT2012171

Certificate of Provisional Acceptance

I do hereby recommend that the thesis work prepared under my supervision by Dhruv Kumar titled "Compressed knowledge transfer via factorization models into recommender systems" be accepted in the partial fulfillment of the requirements of the degree of Bachelor of Technology (B.Tech), in Information Technology at Indian Institute of Information Technology, Allahabad.

Place: Allahabad

Date: 23/07/2016

_____

Dr. Ratna Sanyal
Asst. Professor
IIIT Allahabad

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

FAKULTÄT FÜR WIRTSCHAFTS-WISSENSCHAFTEN

UNIVERSITÄT PADERBORN | 33095 PADERBORN

To

The Evaluation Board
Devghat, Jhalwa
Allahabad-211012
U.P India

Jun.-Prof. Dr.
Artus Krohn-Grimberghe

Lehrstuhl für
Wirtschaftsinformatik
Analytische Informationssysteme und
Business Intelligence

Warburger Str. 100
33098 Paderborn

Raum Q2.454
Fon 0 52 51. 60-23 81
Fax 0 52 51. 60-35 42
E-Mail artus@aisbi.de
Web www.aisbi.de

Sekretariat
Carina Trimborn
Raum Q2.451
Fon 0 52 51. 60-52 45
Fax 0 52 51. 60-35 42
E-Mail trimborn@aisbi.de
Web www.aisbi.de

Certificate of Completion

This is to certify that the research project titled "Compress Knowledge Transfer via Factorization Models in Recommender Systems" has been successfully carried out at the Analytic Information and Business Intelligence (AIS/BI) Lab at Universität Paderborn, Germany by Mr. Dhruv Kumar and Mr. Kumar has successfully completed his internship.

Period of project: 18th Jan 2016 – 10th June 2016

To best of my knowledge and judgement, Mr. Dhruv Kumar's work constitutes not only a mere implementation but original research advancing the state of the art. I'd wholeheartedly like to thank Mr. Kumar for his dedication, skill and hard work. Given the results of his work, I have offered Mr. Kumar a master's student position in my lab.

Paderborn, June 9, 2016

Jun.-Prof. Dr. Artus Krohn-Grimberghe

Acknowledgements

I express my sincere gratitude to Prof. Artus Krohn-Grimberghe, AIS/BI, University of Paderborn, Germany. I am privileged to experience a sustained enthusiasm and interest from his side. Working under him motivated me to be research oriented and try everything before I gave up. His efforts for my research always fueled my interest, and nothing, but for this help I was successfully able to conduct experiments on my research work.

I also extend my sincere thanks to Dr. Ratna Sanyal for the encouragement, motivation and cooperation.

I thank every person whose slightest contribution has made this work possible.

Abstract

In the past years, recommender systems have been used in many web applications to predict products, movies, songs, etc. for the users. The basic task is to figure out the top items that the user might like. In recent years, it has also received a lot of interest from the academia. There has been a lot of interest shown in a collaborative filtering algorithm called Matrix Factorization as it has demonstrated a better performance than the other Collaborative Filtering and Content Based Filtering Algorithms. The former approach looks at the relation between different items and different users and tries to predict on the basis of this similarity, whereas the latter looks at the information (metadata) provided about the items and the users and tries to predict on the basis of this. Thus far the hybrid methods of collaborative filtering and content based approach have not been successful in integrating content based metadata for improving the performance. My goal in this project is to test a hybrid approach and test how well it is able to integrate the metadata in improving the performance of the system and then looking into the behavior of a state-of-the-art model called the factorization machines when the metadata is added to it.

Contents

Chapter 1

Introduction

The users nowadays spend a lot of time online, searching for products and information. Therefore it has become a task of utmost importance to make their searching efficient by helping them with information that is useful to them and eliminate the information that may not be valid for a particular user. For example, YouTube recommends videos to its users based on their previous searches and the videos they had watched previously as well as from the channels one has subscribed. Similarly, movie and TV streaming website Netflix, recommends movies and TV series to its users based on their watching history. If the users don't waste their time on searches that are not informative to them there would be increase in the profit that the providers make.

This is the reason so many of the online retailers are using this technology. It tends to add an extra dimension in the user's experience. If the user has a good experience, there is a chance that he would return to the website. This has therefore become the key to the user's satisfaction and their loyalty towards the service providers. Therefore almost all of the service providers have begun studying and analyzing the user patterns and trying to include this information in predicting items to their users.

1.1 Problem Definition

In my study, I have n users and m items. These users have rated these items and we have these explicit ratings r. We have our data is in the form of user, item, rating <u, m, r> tuples. Beside this we have some additional information in the form of content metadata about the items. We have a short summary about each movie. Further we have the information about the cast of the movie including the actors, producers, director, etc.

1.2 Auxiliary data

There are 2 perspectives to look at the auxiliary data (metadata) that can be used for improving the performance of the recommender systems.

User generated vs expert: There is some information that is given by the user about the product or even himself. This include ratings given by user, reviews written by him and click history. There is also this other category which includes information entered by an expert like a description about the product, relevant tag given and general information about the product.

Content vs Context vs Network vs Feedback: The other perspective is given in [8] which classifies metadata into 4 categories

- Content metadata- User's static profiles, Item's description
- Context metadata- User's dynamic mood, information of time
- Network metadata- Social relations of users, friends
- Feedback metadata- feedback given by the user, browsing history of particular item

For the rest of the paper I am going to follow the second perspective about the metadata.

1.3 Goal

My goal then is to find an efficient way both in terms of time and space measures to use this information to improve the accuracy of recommender systems. In my case the auxiliary data that we would be using is the content metadata about the product. The major challenge is to extract some useful information from this and then to bring it to a form in which it is useful to the models. First I test a way to incorporate the processed information from the metadata into matrix factorization and then I test how factorization machines process the same information.

Chapter 2

Literature Survey

2.1 Previous Strategies

Non-personalized approach

This was the first and the least complicated approach towards recommender systems. Through it the same recommendation is given to all. They take in all the data that they get and generate recommendations like “bestseller”, “trending”, etc. They do not require any personal information regarding the user.

Content Based Approach

This approach requires a lot data that the user may have to manually enter. This data includes information about the user as well as the movies. The information about the user might include demographic information or the user filling a questionnaire. The information about the movie might include explicitly which genre it is, its box office popularity, etc. This model thus builds profiles for users and movies and then tries to match these profiles to predict the movies that a user might like [4]. The issue with this approach is that getting all this information from the user and also about the movie can be a very hard task. One more drawback is that by using this approach the system is restricted to recommending the same type of movies that a user is already watching.

Collaborative Filtering Approach

It is quite a popular recommender system technique which uses the information from other users to predict the rating for a particular user. The underlying intuition that it uses is that we can find similar users and then use similarity between them to find people with similar tastes and use that to predict the movies that a particular user might like. For instance, suppose that a group of users have a similar taste with a person X, then we might suggest this person X the movies which these group of users have rated highly and the user X has not seen them.

There are various types of algorithms within Collaborative Filtering.

Memory Based (Nearest Neighbor)

1. **User-User CF**

This is also known as k-Nearest Neighbor CF. It was first introduced in the GroupLens Usenet article recommender [1]. The basic philosophy is finding the k most similar users and then using this to predict the movies. First a similarity matrix is computed between the users. There could be many ways to calculate this like using Pearson Correlation,

Cosine Similarity, etc. Then the rmse values for the movies for a particular user is calculated using these similarity matrix and finally the top movies based on these rmse values is served to the user. Pearson Correlation has been found to be the best method [2][3]

2. Item-Item CF

Although the User-User CF method gives accurate results but it suffers from a problem of scalability as the number of users grows. To deal this problem this approach is used. In this approach instead of finding the similar users, we find the similar movies. Item–item collaborative filtering was first described in the literature by Sarwar et al. [9] and Karypis [10]. If two movies have been watched by the same group of people and they all tend to feel similar about a movie then the movies are similar. The similarity matrix is calculated in the same way as in the User-User CF method and then the top movies which are similar to the movies that the user has rated highly is served to the user. The main advantage of this method is that once many users have given the ratings for an item then we can pre-compute this similarity matrix and it saves a lot of time. Even when a new user comes and rates a product there would not be much change in the similarity matrix. The similarity can be re-computed in some fixed intervals of time to account for the new ratings or the changed ratings, but the interval would not be that short.

Model Based (Latent Factor Models)

These have proved to be superior to the previous class of Collaborative Filtering algorithms in terms of the predicted accuracy. In this model the factors are these hidden characteristics of both users and movies. These factors are learnt by the system and based on the dot product of these factors, the ratings are predicted.

Hybrid Approach

This approach mainly seeks to combine the advantages of both content based and collaborative filtering approaches. Also models that use different techniques all of them from a same approach could also be called a hybrid model. One way in which the hybrid systems are used are by adding content based capabilities to a collaborative filtering system. We have used this approach in the first part of our research. We tried to incorporate additional information from the metadata to the matrix factorization model.

Context Aware Approaches

In general the recommendation systems are not good in taking into account the extra information like the time, place, weather, etc. when a product was selected. The challenge in adding this type of information is that the feature space increases. Already the recommender systems have problem dealing with the sparsity of the information that we have. When extra dimensions are added the already sparse feature space becomes

sparser and becomes a very big problem. Therefore we need efficient algorithms which can deal with such large sparsity. Models like tensor factorization and factorization machine can handle the above mentioned issues.

2.2 Related Work

A lot of research has been done previously to incorporate different type of metadata to recommender system models earlier. Basically as described in [20] recommender systems are broadly classified into 3 categories which are content based, collaborative filtering and hybrid approach. As we see in [21], [22] creating a hybrid approach by adding content metadata information to matrix factorization methods have not been so successful in the past. In [23] we see how time related metadata can be used in collaborative filtering. In [24] we see how user and product profile are built by using user reviews (feedback) as source of extra information. This is where tensor factorization methods [25], [26] gained popularity as they were able to better handle this extra contextual information by doing factorization in more dimensions. Then Rendle in [13] introduced factorization machines. They include polynomial regression with factorization techniques. They in almost all cases perform better than tensor factorization. They are known to mimic famous factorization models like Matrix Factorization (MF) [8] and Bayesian Probabilistic Matrix Factorization (BPFM) [27]. As shown in [15] they are also better to suited to handle extra contextual information.

In general user generated or collaborative metadata is found to be more useful in improving the performance of recommender systems than say content metadata. In [28] the authors talk about implicit knowledge about user's action of browsing and clicks (feedback metadata) can be used to further enhance the performance of factorization machines. In [30] the authors talk about that training the model not all data but on a right slice of data. The slice is chosen by grouping items based on their properties. In [29] the authors have tried to utilize additional explicit feedback metadata like binary like/dislike ratings through factorization machines. But as we see not much work has been done in analyzing how factorization machines handle content metadata.

2.3 Background

Matrix Factorization

This method comes under the Latent Factor Models of the Collaborative Filtering Algorithm. It has now become the basis of every standard Recommender System machine as it can handle both implicit and explicit data and gives pretty accurate predictions. It was proposed by C Volinsky et al [8]. Matrix factorization technique maps the user and items to a joint latent factor space of a fixed dimension and then the user item

interactions are obtained by interaction between the latent factor spaces. It is the preferred method over SVD which has problems dealing with incomplete matrices. The user-item relation is mapped to a fixed dimension f by singular value decomposition of the ratings data.

The ratings data is seen as two dimensional matrix where one axis is the set of the users (U) and the other is set of the movies (M). The entries in the data is considered as the preference the user has for a movie. If there is no value then the preference is considered to be zero. Let this matrix be called R. The size of R would be $|U|x|M|$. Now to do SVD this matrix can be broken and also regenerated by the dot product of two matrices namely P (U x K) and Q (M x K), where K is the latent factors. R is approximated by the dot product of the two matrices given by the equation

$$R_{ui} = P_u Q_i^T \quad (1)$$

In this way each row of P would represent the strength of the associations between the user and features and similarly each row of Q would represent the strength of the associations between the movie and features. Now our main aim is to map the (user, item, rating) tuples from the training set to the joint latent factor space of dimension f. The problem is that in general users tend to rate very few number of movies. Therefore we have a very sparse matrix in R. Therefore we need to add the regularization term in our equation to prevent overfitting of data. The equation that we need to minimize is

$$\min_{Q,P} \sum_{(u,i) \in K} (e_{ui}^2 + \lambda(||Q_i||^2 + ||P_u||^2)) \quad (2)$$

$$e_{ui} = R_{ui} - P_u Q_i^T \quad (3)$$

λ is the regularization factor. K denotes the rating tuples. The model learns by the fitting the observed ratings.

Learning Algorithms

There are generally two methods used to minimize the above equation

1) **Stochastic Gradient Descent** – The algorithm loops over all the ratings that we have in the matrix R and predicts r_{ui} and thus calculates the error. Then it updates the matrix P and Q accordingly through the following equations. α is the learning rate and β is the regularization factor.

$$Q_i <- Q_i + \alpha (e_{ui} P_u - \beta Q_i) \quad (4)$$

$$P_u <- P_u + \alpha (e_{ui} Q_i - \beta P_u) \quad (5)$$

2) **Alternating Least Squares** - Because both Q_i and P_u are unknowns, minimization equation is not convex. However, if we fix one of them, then the optimization problem becomes quadratic and can be solved optimally. Thus, ALS technique rotates between fixing Q_i and P_u. When Q_i is fixed the system re-computes the P_i's and vice versa.

While in general stochastic gradient descent is easier and faster than ALS, ALS is favorable in at least two cases. The first is when the system can use parallelization. In ALS, the system computes each Q_i independently of the other item factors and computes each P_u independently of the other user factors. This gives rise to potentially massive parallelization of the algorithm. The second case is for systems centered on implicit data. Because the training set cannot be considered sparse, looping over each single training case—as gradient descent does—would not be practical. ALS can efficiently handle such cases

Chapter 3

Requirements

3.1 Python

The whole project is written in Python 2.7.

3.2 Sublime

It is a cross platform source code editor with a Python application programming interface (API). It natively supports many programming language and markup languages, and its functionality can be extended by users with plugins, typically community built and maintained under free licenses. I have used the 3.0 version.

3.3 Numpy

NumPy is an extension to the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays. NumPy is open source and has many contributors. I am using the "1.10.1" version.

3.4 Plotly

Plotly, also known by its URL, Plot.ly, is an online analytics and data visualization tool, headquartered in Montreal, Quebec. Plotly provides online graphing, analytics, and stats tools for individuals and collaboration, as well as scientific graphing libraries for Python, R, MATLAB, Perl, Julia, Arduino, and REST.

3.5 Sci-kit learn

scikit-learn(formerly scikits.learn) is an open source machine learning library for the Python programming language. Scikit-learn integrates machine learning algorithms in the tightly-knit scientific python world, building upon numpy, scipy, and matplotlib. As a machine-learning module, it provides versatile tools for data mining and analysis in any field of science and engineering. It strives to be simple and efficient, accessible to everybody, and reusable in various contexts.

3.6 Libfm

It is the most state-of-the-art library which gives us the implementation of factorization machines. It is written by Rendle himself and is fast and scalable. It gives us four models as implementation. They are Alternating Least Square (ALS), Stochastic Gradient Descent (SGD), Adaptive Stochastic Gradient Descent (SGDA) and Markov Chain Monte Carlo

(MCMC). We have used this software implementation to study the behavior of factorization machines on metadata and determine the importance of metadata preprocessing.

Chapter 4

Proposed Approach

As explained in section 2.3, I have chosen the Matrix Factorization as the base of my approach. There also have been few proposed approaches for this like [11]. The basic purpose of the following algorithm will be to try to include metadata into the Matrix Factorization and see if it helps us to improve the accuracy even further.

4.1 Type of data

I have used MovieLens 1M dataset for the ratings of the movies given by the user. It has 1 million tuples (user, movie, ratings). It has 6041 users and 3953 movies.

I have taken the metadata from DBpedia. It has abstracts of the movies and also has the directors, producers, actors and writers in the movie.

4.2 Algorithm

4.2.1 Adding Genre Information

The first task is to find out something from the abstract that could be useful for our Matrix Factorization approach. I have tried to figure out the genres of the movies through the abstract given. The most basic approach could be to get a list of genres and then search for these keywords in the abstract. But this approach has certain shortcomings. For example, if a different form of the word (genre) is present in the abstract it would be missed. For example, if the genre of a movie is "History" and the word "historical" appears in the abstract, it would be missed. Therefore, I have stemmed the genres and brought them to their root form, so even if there is a word with a different form it can get labelled. Also I have added certain keywords in the genre list like "award". The reason behind this is some users are affected by whether a movie was in the contention to win an award. After this step is implemented we get a MG (movie x genre) matrix. This is a binary matrix which tells whether a particular movie has characteristics of a particular genre or not. We then normalize this matrix by dividing the values by $\sum g_m$ which represents the total number of genres in a movie. This step is necessary as there may be more than one genres of a movie and the user may like a movie more because of a particular genre.

The next step is to get a UG(user x genre) matrix from this. The formula to calculate a particular cell value in the UG matrix is

$$UG_{ij} = \sum r_i g_i / n_j \quad (6)$$

where i is the movie number, j is the genre number and n is the number of movies which had genre j in them. The table 1 shows this matrix.

User1	G1	G2	G3	Rating
M1	0.5	0	0.5	5
M2	0	1	0	2
M3	0.33	0.33	0.33	4
UG	1.91	1.66	1.91	

Table 1: Calculation of UG matrix

In addition to minimizing equation 2 we also need to minimize the following equation and update accordingly. The error equation would be similar to equation (3).

$$\min_{G,P} \sum_{(u,g)\,\epsilon\,K} (e_{ug}^2 + \lambda(||G_g||^2 + ||P_u||^2)) \quad (7)$$

$$e_{ug} = UG_{ug} - P_u G_g^T \quad (8)$$

As mentioned above there would be two new update equations in addition to the equations (4) and (5). This would add the effects of the inclusion of genre information in matrix factorization.

$$G_g <- G_g + \gamma (e_{ug} P_u - \beta G_g) \quad (9)$$
$$P_u <- P_u + \gamma (e_{ug} G_g - \beta P_u) \quad (10)$$

4.2.2 Adding Cast Information

In this part I tried to incorporate the information of the actors, producers and writers in the movies to the algorithm.

4.2.2.1 Dimensionality Reduction Approach

First I tried adding the information in pretty much the same way as the genre information. The main difference is that in the case of genres the matrix (MG) size was very less as the size of the genres list was 23, but in this case the size of matrix (MA) is very large as the number of actors, producers, etc. is 1982 (actor count). Therefore, it is very

computationally intensive. To solve this problem, I applied dimensionality reduction over this matrix (MA) to reduce the dimension to 100. Then the matrix UA (user x actors) is calculated in the same way as the matrix UG was calculated previously through the equation (6).

4.2.2.2 Clustering Approach

In approach 1, we applied dimensionality reduction, but one issue that it faces is that some information is lost. Therefore through this second approach I propose an algorithm through which no information is lost and we can reduce the complexity that we face. I have opted to use clustering for condensing the information as it does not cause any loss of information. So now initially we have the MA matrix which is very large in size. Then I calculate a matrix AG (actor x genre) which basically tells us what is the preference of the actors, directors, producers, etc. with respect to the genres. Then through this matrix I calculate a similarity matrix between actors, simA. The measure I used to calculate the similarity was cosine. The basis was the genre. The cosine similarity can be put as in equation 11

$$\text{Similarity} = \cos(\Theta) = A.B \,/\, ||A||.||B|| = \sum A_i B_i \,/\, (\sqrt{\sum A_i} \,.\sqrt{\sum B_i}) \qquad (11)$$

Now clustering can be applied on this matrix to group similar actors, directors, producers, etc. together. There are many algorithms that could be applied for clustering. I tried with the following two:

1. **Agglomerative clustering**: Part of the hierarchical clustering strategy, it is a bottom up approach. Each observation starts in its own cluster and pairs of clusters are merged as one moves up the hierarchy. In general, the merges and splits are determined in a greedy manner. The results are usually represented in a dendrogram. I used the scikit learn agglomerative clustering algorithm for implementation.

2. **K-means**: The KMeans algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares. The k-means algorithm divides a set of N samples X into K disjoint clusters C, each described by the mean μ_j, of the samples in the cluster. The means are commonly called the cluster "centroids". The K-means algorithm aims to choose centroids that minimize the *inertia*, or within-cluster sum of squared criterion:

$$\sum_{i=0}^{n} \min_{\mu_j \in C}(||x_j - \mu_i||^2) \quad (12)$$

Now as we have got the number of actors reduced to a manageable account we can now form the MA (movie x actor) matrix as we did earlier and then through it finally the UA (user x actor) matrix which would be used to train our recommender system.

After getting the information from the metadata in the useable form we can formulate the new equation which needs to be minimized as the following

$$\min_{A,P} \sum_{(u,a) \in K} (e_{ua}^2 + \lambda(||A_a||^2 + ||P_u||^2)) \quad (13)$$

$$e_{ua} = UA_{ua} - P_u A_a^T \quad (14)$$

The final update equations to be added to the algorithm are as follows

$$A_a <- A_a + \delta (e_{ua} P_u - \beta A_a) \quad (15)$$

$$P_u <- P_u + \delta (e_{ua} A_a - \beta P_u) \quad (16)$$

4.2.3 Time Complexity Analysis

In this section I have compared the time complexity of the proposed algorithm to matrix factorization. There has been a lot of study done on the non-negative matrix factorization. One such paper like [12] tells us that (i) it is equivalent to a problem in polyhedral combinatorics (ii) it is NP-hard and (iii) that a polynomial time local search heuristics exist.

In simple terms the complexity of the matrix factorization can be written as $\Theta(n^4)$. This can be explained as the product of the number of steps in SGD (500), the number of users (6041), the number of movies (3953) and the count of latent factors (5). But the code runs not for all the user item pair but only for which the ratings are given.

When the equations are modified after adding the genre information, the complexity becomes $\Theta(n^4) + \Theta(m^4)$. The first part is the same as above. The second part can be explained as the product of the number of steps in SGD (500), the number of users (6041), the number of genres (23) and the count of latent factors (5). As we see the second part is 0.006 times the first part. This is a small fraction of the second part.

When the last part is added i.e. the actor information, the complexity becomes $\Theta(n^4) + \Theta(m^4) + \Theta(a^4)$. The added part can be explained as the product of the number of steps in SGD (500), the number of users (6041), the number of actors (100) and the count of latent factors (5). As we see this again is small part (0.03 times) of the original matrix factorization.

But the main difference here is that the additional approaches run for all user item pairs. The speed of matrix factorization depends on the sparsity of the matrix. But we can say that adding the above approaches increases the running time only by a fraction of the original running time of matrix factorization.

Chapter 5

Factorization Machine

Factorization Machines [13] are a new model class that combine the advantages of SVMs with factorization models. Like SVMs, FMs are general model predictors working with any real valued feature vector. In contrast to SVMs, FMs models all interactions between variables using factorized parameters. Thus they are able to estimate interactions even in problems with huge sparsity (like recommender systems) where SVMs fail.

There are many factorization models like matrix factorization, parallel factor analysis or specialized models like SVD++, PITF or FPMC. The drawback of these models are that they are not applicable for general prediction tasks but work only with special input data. Furthermore their models equations and optimization algorithms are derived individually for each task. FM can mimic these models just by specifying the input data.

The FMs can be compared to SVMs with a polynomial kernel. SVMs define a multidimensional hyperplane, which learns the shape of the curve of the data. However, SVMs have certain weaknesses which are addressed by the FMs. For example, the SVMs do not work well on sparse data. Further, in SVMs, the input variables are still independent variables even though the polynomial kernel attempts to model the interaction among the variables. This takes polynomial time to compute.

The advantages that FMs have over the SVMs are the following:

1. They allow parameter estimation under very sparse data. In fact, they are built specifically for sparse data. They do not perform well on dense data.
2. They have linear complexity, and can be optimized in the primal and do not rely on support vectors like SVMs.
3. They can model n-way variable interactions, where n is the number of polynomial order. Although in general the value of n is kept 2 in most cases.

There are several implementations of factorization machines. The state-of-the-art continues to be libFM [14]. In the original paper [13], Rendle discussed a method of optimization for the model parameters known as stochastic gradient descent, which works well with several loss functions. However, the optimization algorithm is extremely dependent on the learning rate, one of the hyper-parameters of the method. If the learning rate is too high, the model parameters will not converge, while if it is too low, the algorithm is no longer time-efficient.

Because of this, Rendle reviewed three more methods known as alternating least-squares [15], markov chain monte carlo inference [16] and adaptive stochastic gradient descent

[17]. He recommends markov chain monte carlo inference because there are fewer hyper-parameters, and those that must be specified are not as sensitive to their initial values.

The most common prediction task is to estimate a function $y:R^n \rightarrow T$ from a real valued feature vector $x \in R^n$ to a target domain T (e.g. T = R for regression or T = {+, −} for classification). In supervised settings, it is assumed that there is a training dataset D = {(x(1), y(1)), (x(2), y(2)), . . .} of examples for the target function y given. We also investigate the ranking task where the function y with target T = R can be used to score feature vectors x and sort them according to their score. Scoring functions can be learned with pairwise training data [18], where a feature tuple (x(A), x(B)) ∈ D means that x(A) should be ranked higher than x(B). As the pair wise ranking relation is antisymmetric, it is sufficient to use only positive training instances.

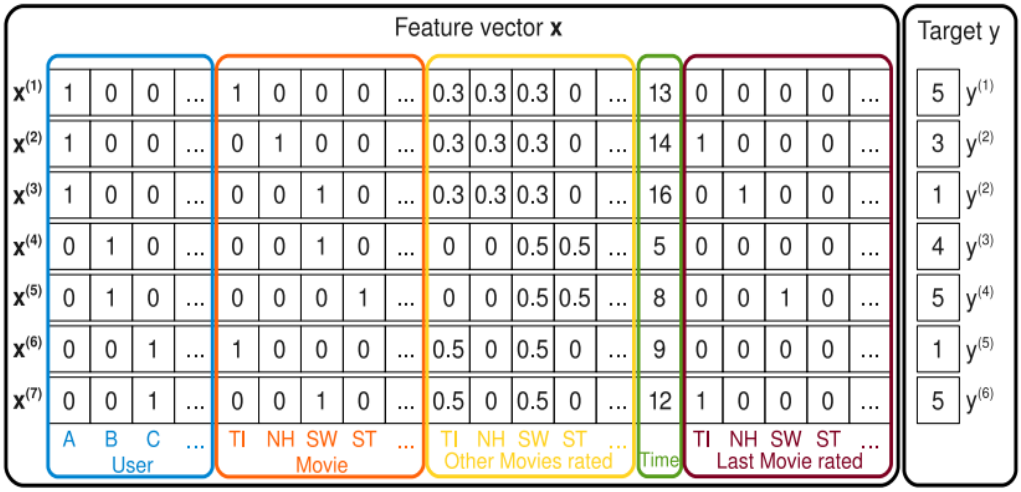

Figure 1: Factorization Machine representation

Assume we have the transaction data of a movie review system. The system records which user u 2 U rates a movie (item) i ∈ I at a certain time t ∈ R with a rating r ∈ {1, 2, 3, 4, 5}. Let the users U and items I be:

U = {Alice (A), Bob (B), Charlie (C), . . .}
I = {Titanic (TI), Notting Hill (NH), Star Wars (SW), Star Trek (ST), . . .}

Let the observed data S be:

S = {(A, TI, 2010-1, 5), (A,NH, 2010-2, 3), (A, SW, 2010-4, 1),
(B, SW, 2009-5, 4), (B, ST, 2009-8, 5),
(C, TI, 2009-9, 1), (C, SW, 2009-12, 5)}

An example for a prediction task using this data, is to estimate a function y that predicts the rating behavior of a user for an item at a certain point in time. Figure 1 shows one example of how feature vectors can be created from S for this task. Here, first there are |U| binary indicator variables (blue) that represent the active user of a transaction – there is always exactly one active user in each transaction (u, i, t, r) ϵ S, e.g. user *Alice* in the first one (x(1) A = 1). The next |I| binary indicator variables (red) hold the active item – again there is always exactly one active item
(e.g. x(1) TI = 1). The feature vectors in figure 1 also contain indicator variables (yellow) for all the other movies the user has ever rated. For each user, the variables are normalized such that they sum up to 1. E.g. *Alice* has rated *Titanic*, *Notting Hill* and *Star Wars*. Additionally the example contains a variable (green) holding the time in months starting from January, 2009. And finally the vector contains information of the last movie (brown) the user has rated before (s)he rated the active one – e.g. for x(2), *Alice* rated *Titanic* before she rated *Notting Hill*. In section V, we show how factorization machines using such feature vectors as input data are related to specialized state-of-the-art factorization models.

The model equation for the FMs is the following

$$\hat{y}(\mathbf{x}) := w_0 + \sum_{i=1}^{n} w_i \, x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle \, x_i \, x_j \tag{17}$$

where the model parameters that are to be estimated are the following

$$w_0 \in R, \; w \in R^n \text{ and } V \in R^{n \times k}$$

and <v, v> represents the dot product of two vectors

$$\langle \mathbf{v}_i, \mathbf{v}_j \rangle := \sum_{f=1}^{k} v_{i,f} \cdot v_{j,f} \tag{18}$$

A row v_i within V describes the i-th variable with k factors. k ϵ N is a hyper-parameter that defines the dimensionality of the factorization.
A 2-way FM (degree d = 2) captures all single and pairwise interactions between variables:
- w_0 is the global bias.
- w_i models the strength of the i-th variable.
- $w_{i,j} := \langle v_i, v_j \rangle$ models the interaction between the i-th and j-th variable. Instead of using an own model parameter $w_{i,j} \in R$ for each interaction, the FM models the interaction by factorizing it. We will see later on, that this is the key point which allows high quality parameter estimates of higher-order interactions (d >=2) under sparsity.

Now let us talk about the one of the most powerful things about the factorization machines i.e. it's computation complexity. The complexity of straight forward computation of eq. (17) is in $O(kn^2)$ because all pairwise interactions have to be computed. But with reformulating it drops to linear runtime. Due to the factorization of pairwise interaction, there is no model parameter that directly depends on two variables (e.g. a parameter with an index (i, j)). So the pairwise interactions can reformulated.

$$
\begin{aligned}
&\sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle \, x_i \, x_j \\
=&\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle \, x_i \, x_j - \frac{1}{2} \sum_{i=1}^{n} \langle \mathbf{v}_i, \mathbf{v}_i \rangle \, x_i \, x_i \\
=&\frac{1}{2} \left(\sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{f=1}^{k} v_{i,f} \, v_{j,f} \, x_i \, x_j - \sum_{i=1}^{n} \sum_{f=1}^{k} v_{i,f} \, v_{i,f} \, x_i \, x_i \right) \\
=&\frac{1}{2} \sum_{f=1}^{k} \left(\left(\sum_{i=1}^{n} v_{i,f} \, x_i \right) \left(\sum_{j=1}^{n} v_{j,f} \, x_j \right) - \sum_{i=1}^{n} v_{i,f}^2 \, x_i^2 \right) \\
=&\frac{1}{2} \sum_{f=1}^{k} \left(\left(\sum_{i=1}^{n} v_{i,f} \, x_i \right)^2 - \sum_{i=1}^{n} v_{i,f}^2 \, x_i^2 \right)
\end{aligned}
$$

This equation has only linear complexity in both k and n – i.e. its computation is in O(kn).

5.1 FMS VS SVMS

As we mentioned earlier the various advantages that FMs have over the SVMs. Let us now look into some proof regarding the same. We look into the two types of kernels of the SVMs.

- **Linear Kernel**
  The most simple kernel is the linear kernel: $K_l(x, z) := 1 + \langle x, z \rangle$, which corresponds to the mapping $\phi(x) := (1, x_1, \ldots, x_n)$. And thus the model equation of a linear SVM can be rewritten as:

$$\hat{y}(\mathbf{x}) = w_0 + \sum_{i=1}^{n} w_i \, x_i, \quad w_0 \in \mathbb{R}, \quad \mathbf{w} \in \mathbb{R}^n \tag{19}$$

  As you can see that this similar to Factorization Machine of degree of interaction = 1.

- **Polynomial Kernel**

This allows the SVM more interactions between the parameters. It is defined as $K(x, z) := (\langle x, z \rangle + 1)^d$. The equation can be written as follows:

$$\hat{y}(\mathbf{x}) = w_0 + \sqrt{2}\sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} w_{i,i}^{(2)} x_i^2 + \sqrt{2}\sum_{i=1}^{n}\sum_{j=i+1}^{n} w_{i,j}^{(2)} x_i x_j \quad (20)$$

where the model parameters are: $w_0 \in R$, $w \in R_n$, $W(2) \in R_{nxn}$ (symmetric matrix) Comparing a polynomial SVM to a FM, one can see that both model all nested interactions up to degree d = 2. The main difference between SVMs and FMs is the parametrization: all interaction parameters $w_{i,j}$ of SVMs are completely independent, e.g. $w_{i,j}$ and $w_{i,l}$. In contrast to this the interaction parameters of FMs are factorized and thus $\langle v_i, v_j \rangle$ and $\langle v_i, v_l \rangle$ depend on each other as they overlap and share parameters.

First of all, w_u and $w^{(2)}_{u,u}$ express the same – i.e. one can drop one of them (e.g. $w^{(2)}_{u,u}$). Now the model equation is the same as for the linear case but with an additional user-item interaction $w^{(2)}_{u,u}$. In typical collaborative filtering (CF) problems, for each interaction parameter $w^{(2)}_{u,i}$ there is at most one observation (u, i) in the training data and for cases (u' , i') in the test data there are usually no observations at all in the training data. For example in figure 1 there is just one observation for the interaction (Alice, Titanic) and non for the interaction (Alice, Star Trek). That means the maximum margin solution for the interaction parameters $w^{(2)}_{u,i}$ for all test cases (u, i) are 0 (e.g. $w^{(2)}_{A,ST} = 0$). And thus the polynomial SVM can make no use of any 2-way interaction for predicting test examples; so the polynomial SVM only relies on the user and item biases and cannot provide better estimations than a linear SVM.

So now we get a basic overview of factorization machines. Now we will get in depth into the different models through which FMs work.

5.2 STOCHASTIC GRADIENT DESCENT

The model parameters (w_0, w and V) can be learnt through the following method, for a variety of losses like square, hinge, etc.

$$\frac{\partial}{\partial\theta}\hat{y}(\mathbf{x}) = \begin{cases} 1, & \text{if } \theta \text{ is } w_0 \\ x_i, & \text{if } \theta \text{ is } w_i \\ x_i \sum_{j=1}^{n} v_{j,f} x_j - v_{i,f} x_i^2, & \text{if } \theta \text{ is } v_{i,f} \end{cases} \quad (21)$$

The sum $\sum_{j=1-n} v_{j,f}\, x_j$ is independent of i and thus can be precomputed (e.g. when computing y(x)). In general, each gradient can be computed in constant time O(1). And all parameter updates for a case (x, y) can be done in O(kn) – or O(km(x)) under sparsity.

5.3 ALTERNATING LEAST SQUARES

This is the second method, by which the factorization machines are implemented. Before we derive a formulation for this method there are two lemmas that need to be discussed.

- **Lemma 1**: A FM is a linear function with respect to every single model parameter $\theta \in \Theta$ and thus can be represented as:

  $$y(x|\theta) = g_{(\theta)}(x) + \theta h_{(\theta)}(x) \qquad (22)$$

  where $g_{(\theta)}$ and $h_{(\theta)}$ are independent of the value of the parameter θ.

- **Lemma 2**: The regularized least-square solution of a single parameter θ for a linear model y(x|θ) is:

  $$\theta = -\frac{\sum_{(\mathbf{x},y)\in S} \left(g_{(\theta)}(\mathbf{x}) - y\right) h_{(\theta)}(\mathbf{x})}{\sum_{(\mathbf{x},y)\in S} h^2_{(\theta)}(\mathbf{x}) + \lambda_{(\theta)}} \qquad (23)$$

From lemma 2 we see that we can find the optimal value of a parameter directly, given that we have all the remaining parameters with us. A joint optimum can be reached by iteratively calculating the optimum of each parameter one by one and repeating this several times. First the model parameters are initialized, where the 0 and 1-way interactions can be initialized with 0 and the factorization parameters with small 0-centered random values. In the main loop the parameters are optimized one after the other. The idea here is to optimize first lower interactions and then higher ones because for lower interactions more data is observed and thus their estimates are more reliable. Within the factors of the 2-way interactions, first all features of the first factor dimension are optimized, then the features of the second factor dimension, etc. This allows the f-th factor to find the residuals for the 1st to (f − 1)th factor dimensions. This optimization main loop us repeated several times to converge to the joint optimum of all model parameters.

5.4 MARKOV CHAIN MONTE CARLO

This method tries to mimic the Bayesian Probabilistic Matrix Factorization (BPMF) method. This means that we add a Bayesian inference over the probabilistic matrix factorization. This means that we set hierarchical priors over the hyper-parameters to regularize them. The intuition behind this is that additional (hierarchical) structure put on model parameters shrinks them towards each other as long as no evidence in the data clearly indicates the opposite. The figure below compares the probabilistic interpretation of standard Factorization Machines to Bayesian Factorization Machines.

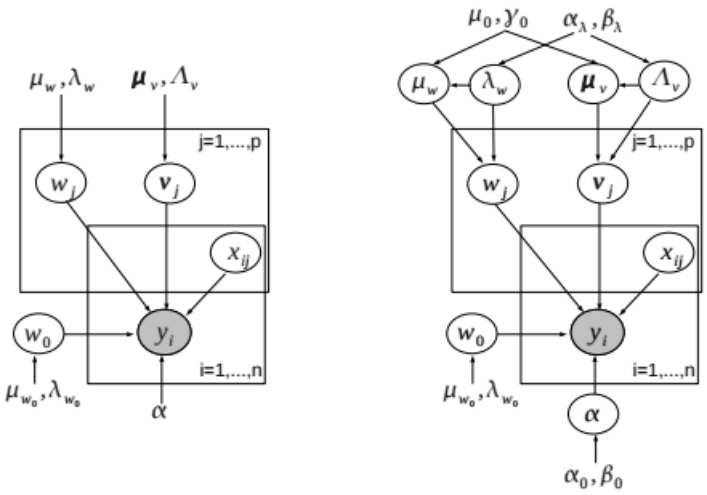

Fig 2: PMF vs BPFM

An example where this helps could be when we need to find the number of latent factors. Typically, the number of latent dimensions k for latent variable models is not known. Too small k under fit, too high k over fit the data. Structured Bayesian methods using hierarchical priors overcome brute force grid search for k. The samples $\{U_i^{(k)}, V_j^{(k)}\}$ are generated by running a Markov chain whose stationary distribution is the posterior distribution over the model parameters and hyper-parameters.

Chapter 6

Experiment testing and analysis

6.1 Data splitting

To calculate the average rmse I have take the mean value of the calculated rmse's from different cross-validation splits. 5 fold 80-20 cross validation split has been used. There were some inconsistencies with the movies in the LastFm dataset and MovieLens dataset. So only those movies were taken for which the additional metadata information was given. After reduction the total number of movies left were 3274. The tuples which contained the movie whose abstract was not given were removed.

Finally after running the algorithm on different cross validation runs the mean rmse was calculated.

$$\text{Mean RMSE} = (\text{RMSE1} + \text{RMSE2} + \text{RMSE3} + \text{RMSE4} + \text{RMSE5}) / 5 \quad (24)$$

6.2 Hyper-parameters

There are generally 3 most common methods to tune hyper-parameters

1) **Grid Search**: It picks out a grid of values and then evaluates every value and then returns the value which gives the best result. Some guesswork is necessary to aim the algorithm to search in the right direction. It is the most expensive method in terms of computation time but we can reduce it by parallelizing it.
2) **Random Search**: It is a simple solution which is surprisingly effective. It is a slight variation of grid search. Instead of searching at every point in the grid, it searches only at some random points in the grid. This makes it vary cheaper in terms of computation time. As it does not searches the entire grid therefore it does not beat grid search. But as shown by Bergstra and Bengio[7] random search performs the same as gird search. All in all trying 60 random points sampled from the grid should be good enough.
3) **Smart Hyper-parameter tuning**: Instead of preparing a batch to search initially, it picks some points, evaluates them and then decides on the basis of the evaluation the next points to search. Therefore it cannot be parallelized. There is a lot complexity in implementing these smart algorithms and making them run faster than random search. They have their own hyper-parameters that need to be tuned to run efficiently.

I have used grid search to tune the hyper-parameters initially using some the standard values to start and then building on that as I went ahead.

The following table shows the value of tuned hyper-parameters for the hybrid matrix factorization.

Hyper-parameters	Value
Alpha	0.004
Beta	0.06
Gamma	0.00055
Delta	0.00075
LF	8

Table 2: Values of hyper-parameters giving optimal results

The LF has been set to 8 as a standard value.

The following graphs also shows the rmse values for different values of gamma and delta.

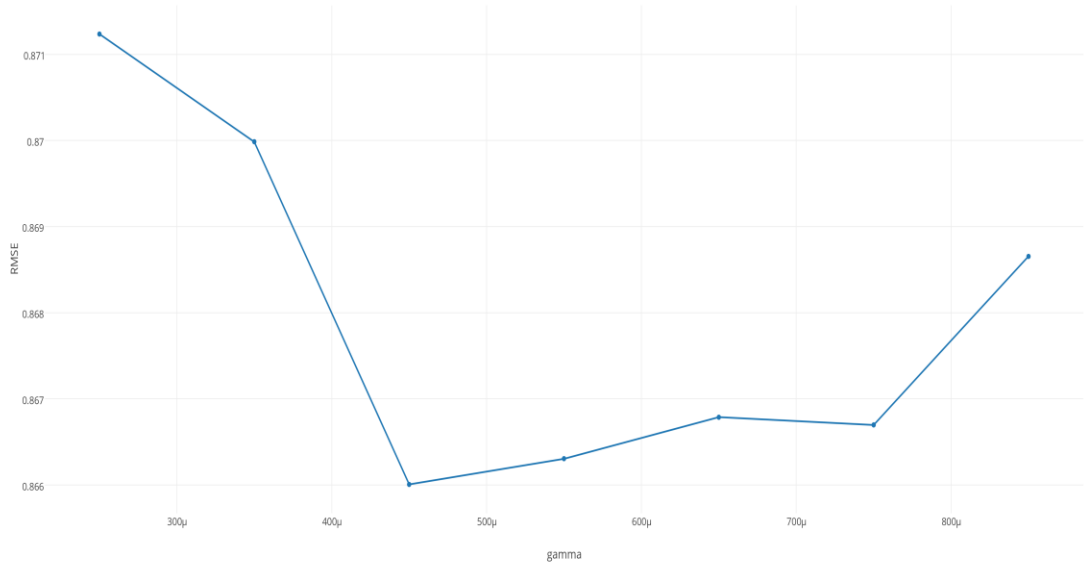

Fig. 3: Gamma vs RMSE

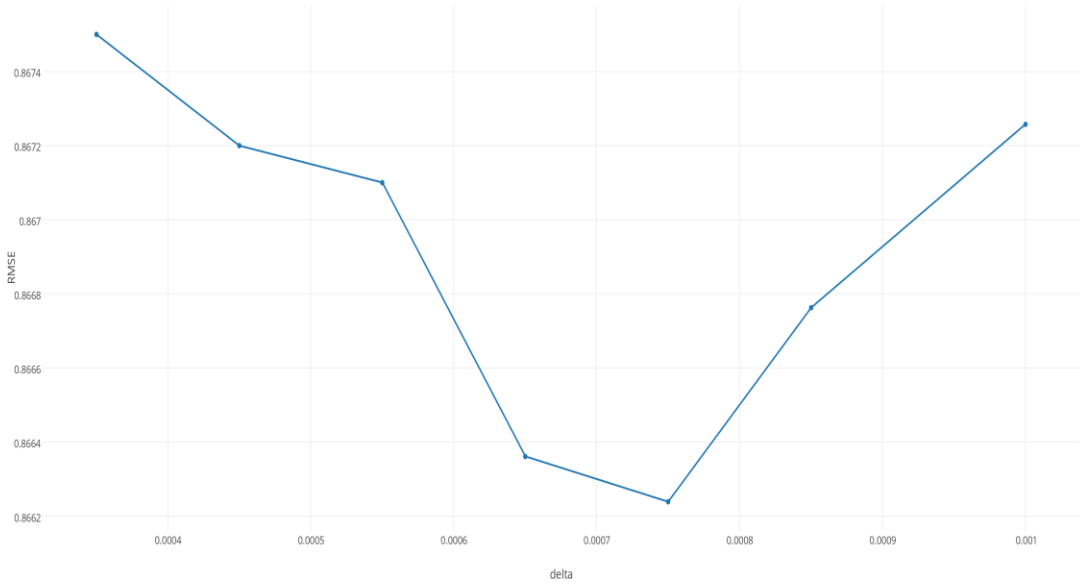

Fig. 4: Delta vs RMSE

Now I look at the values of the hyper-parameters that I needed to tune for different models that can be implemented through the factorization machines.

- SGD
  Learning rate- 0.003
  Regularization rate for 2-way interactions- 0.06
  Number of latent factors for each parameter- 8

- ALS
  Regularization rate for 2-way interactions- 0.06
  Number of latent factors for each parameter- 5

- MCMC
  Regularization rate for 2-way interactions- 0.06
  Number of latent factors for each parameter- 12

6.3 Experimental Results

6.3.1 Baseline Prediction

First to have a baseline for comparison, I evaluated the following algorithm.

Let b_{ui} denote the baseline prediction of user u for item i. The simplest method could be to predict the average overall ratings in the system: $b_{ui} = \mu$(mean). This can be enhanced by further calculating the average rating for user (b_u) and item (b_i). The final enhancement that can be done is to combine them all to predict the rating. The formula for this is as follows

$$b_{ui} = \mu + (b_u - \mu) + (b_i - \mu) \quad (25)$$

The intuition for this is as follows. Suppose we want an estimate for a user X's rating of the movie M. Let's say that the average rating over all movies is μ, is 3.7 stars. Furthermore movie M is better than the average movie, so it tends to be rated 0.7 stars over average. Furthermore, user X is a critical user and tends to rate 0.4 star below the average. Thus the estimate of rating of movie M for user X would be (3.7 + 0.7 – 0.4).

The rmse that I got by this approach was **1.07**.

6.3.2 Hybrid Matrix Factorization

The comparison is made between the MF with the proposed algorithm which includes information from the metadata as well. This comparison is shown in table 3 which has the mean rmse values given by the 4 different algorithms. MF represents the basic matrix factorization, HMFG represents hybrid matrix factorization with genre metadata, HMFC1 represents hybrid matrix factorization with cast information (dimensionality reduction) metadata, HMFC2 represents hybrid matrix factorization with cast information (clustering) metadata, and HMFGC represents hybrid matrix factorization with cast and genre information metadata.

As we see from the table, adding genre information improves the RMSE. Out of the two approaches to add cast information, the clustering approach outperforms the dimensionality reduction approach. This is as a lot more information is lost when we apply dimensionality reduction. Adding all the information further improves the result. Also we have fig. 5 and 6 which serves as a visual tool for comparison between different algorithms. This visual tool helps us to visualize the different rates at which these different algorithm converges and then we can use the most appropriate approach.

Method	Mean Rmse
MF	0.8598
HMFG	0.8585
HMFC1	0.8594
HMFC2	0.8577
HMFGC	0.8579

Table 3: Rmse value for different metadata in HMF

As we see in fig. 5, the clustering approach is much better than the dimensionality reduction approach from the start. It converges quicker. The reason for this is that a lot more information is lost when we apply dimensionality reduction than that in clustering.

As we see from fig. 6, the MFG algorithm is the fastest algorithm in terms of the convergence rate initially. But finally the MFA_cluster beats it in the RMSE values. Thus we see that our hybrid MF model with content metadata added to it performs better than the collaborative matrix factorization. But the important thing to see is that the improvement is very little. This matches with the previous research that has taken place in this field of adding content metadata to MF method. To overcome this as discussed above other methods multiverse tensor factorization and factorization machines have been used. In the next section we see the results that we got when we added the same content metadata to the factorization machines.

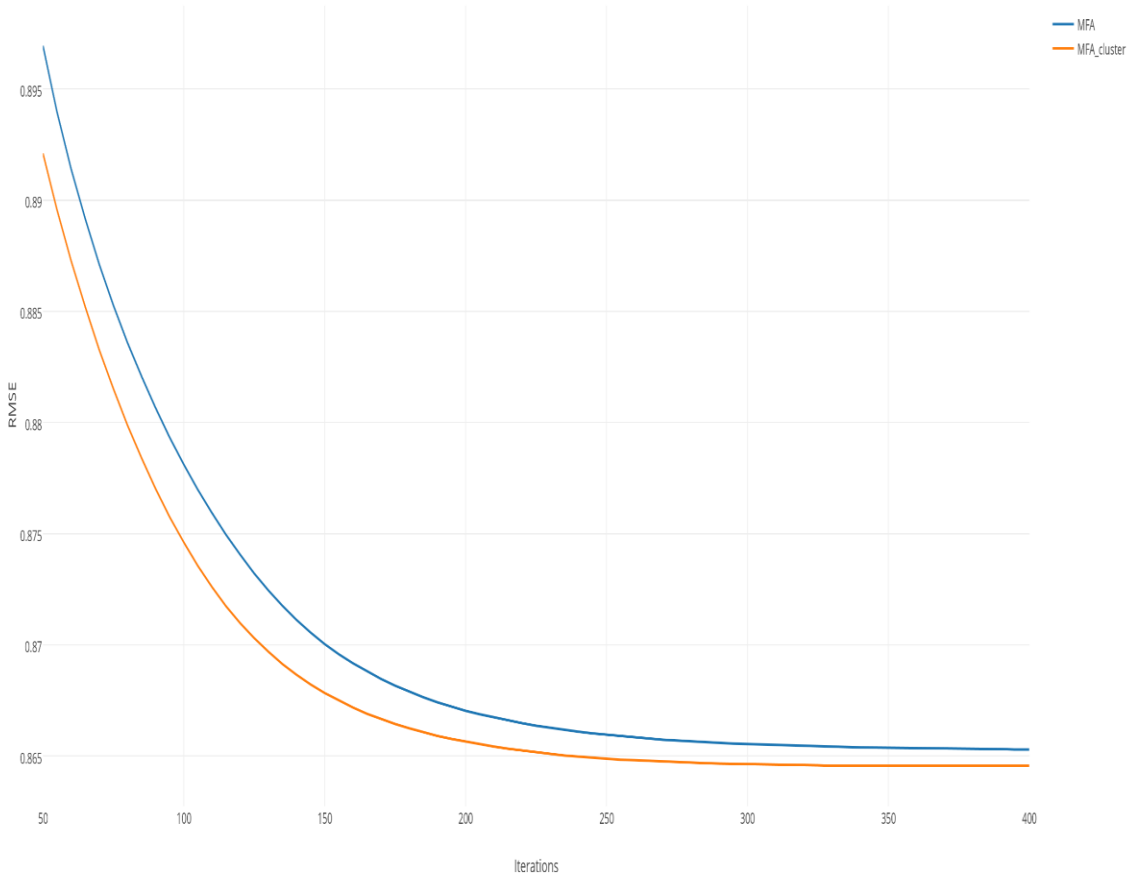

Fig. 5: Comparison of two methods of adding cast metadata

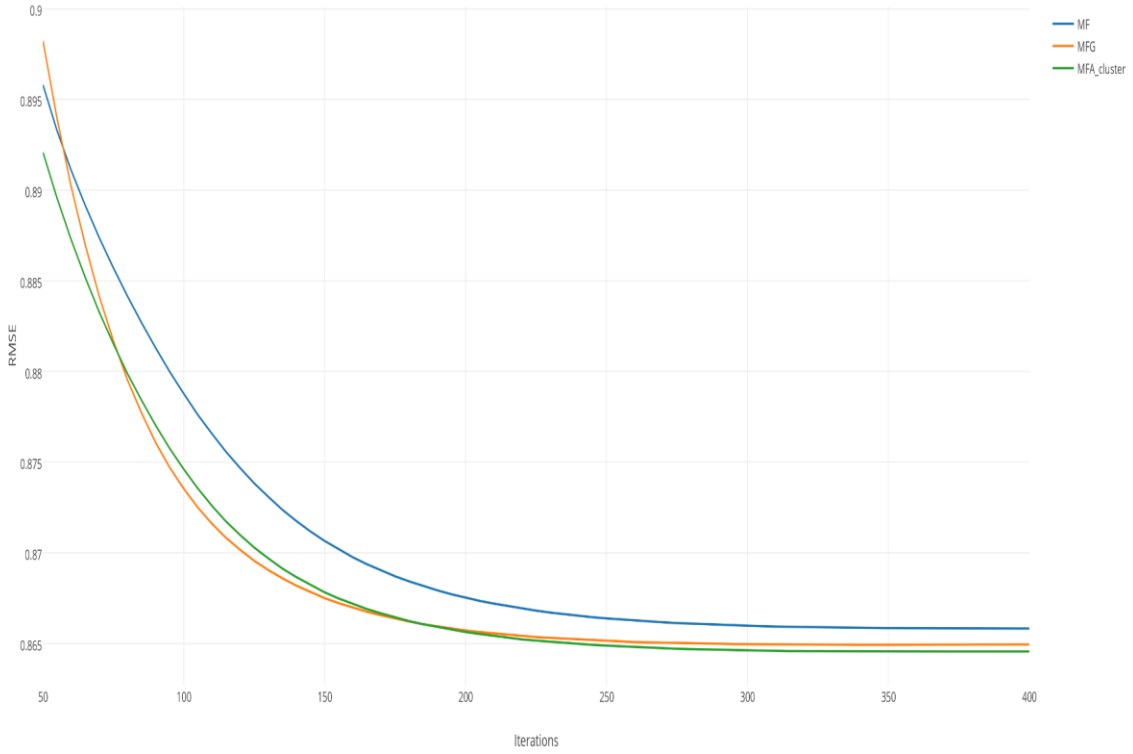

Fig. 6: Comparison of adding metadata with no metadata model

6.3.3 Factorization Machine with Metadata

The same data was added to the factorization machines as to the hybrid matrix factorization.

The results are computed for the different approaches through which factorization machines can be implemented and with all the different metadata that can be added to them. As approach 1 for adding the cast metadata was not effective therefore in this case we only consider the approach 2 in our results.

The following tables show the results for different approaches and with different information.

ALS	RMSE
Without Metadata	0.8681
Actor (Method 2 Metadata)	0.8653
Genre Metadata	0.8648
Actor – Genre Metadata	0.8628

Table 4: Rmse value for ALS

SGD	RMSE
Without Metadata	0.8571
Actor (Method 2 Metadata)	0.8567
Genre Metadata	0.8539
Actor – Genre Metadata	0.8526

Table 5: Rmse value for SGD

MCMC	RMSE
Without Metadata	0.8419
Actor (Method 2 Metadata)	0.8404
Genre Metadata	0.8375
Actor – Genre Metadata	0.8357

Table 6: Rmse value for MCMC

The first thing to note that the improvement that we got after adding metadata to factorization machines was much more that the improvement we got after adding metadata to matrix factorization. The SGD approach here is what resembles that matrix factorization that we implemented earlier. The thing to note here is that the rmse value is almost the same (0.8598 for the raw data earlier and 0.8571 when implemented through factorization machine). Therefore, we can conclude that factorization machines are good at mimicking the models.

As we see in the improvements are quite significant as compared to the improvements we got through matrix factorization. Table 7 shows the percentage improvement that we got through different approaches. This percentage is calculated by the following formula

$$\text{Percentage change} = (Y - X) / Y \quad (26)$$

where Y -> Rmse value with base data and X -> Rmse value with metadata incorporated

Method	**Maximum Percent Change in Rmse**
HMF	0.244
FM - ALS	0.611
FM - SGD	0.525
FM - MCMC	0.736

Table 7: Maximum percentage change

As we see the percent change for the MF method which we discussed earlier is much less than the SGD, although both methods are essentially the same and give the same rmse values on the base data, but behave very differently when metadata is added to it.

When we compare the different algorithms that were implemented through factorization machine, MCMC performs the best, followed by SGD and then by ALS. Also in terms of percentage change, the most change is recorded in MCMC, followed by ALS and then by SGD. In terms of time and iterations taken to converge, ALS is the fastest followed by SGD and MCMC. The figure 6 shows this point. The ALS algorithm converges in about 40-50 iterations, SGD takes about 200-250 iterations and MCMC takes about 500-600 iterations to converge.

The figures shown below gives us a visual representation of the different models. Fig. 6 is the basic comparison of the 3 algorithms implemented through the factorization machine. It gives us an understanding towards us how the 3 algorithms behave in terms of rmse and the convergence rate on the base data. We are able to see that MCMC is the best of the three. Fig. 7, 8 and 9 show how the three algorithms behave when different pre-processed metadata is added to them. The data when all the information from the metadata is added gives us the best result in all the cases.

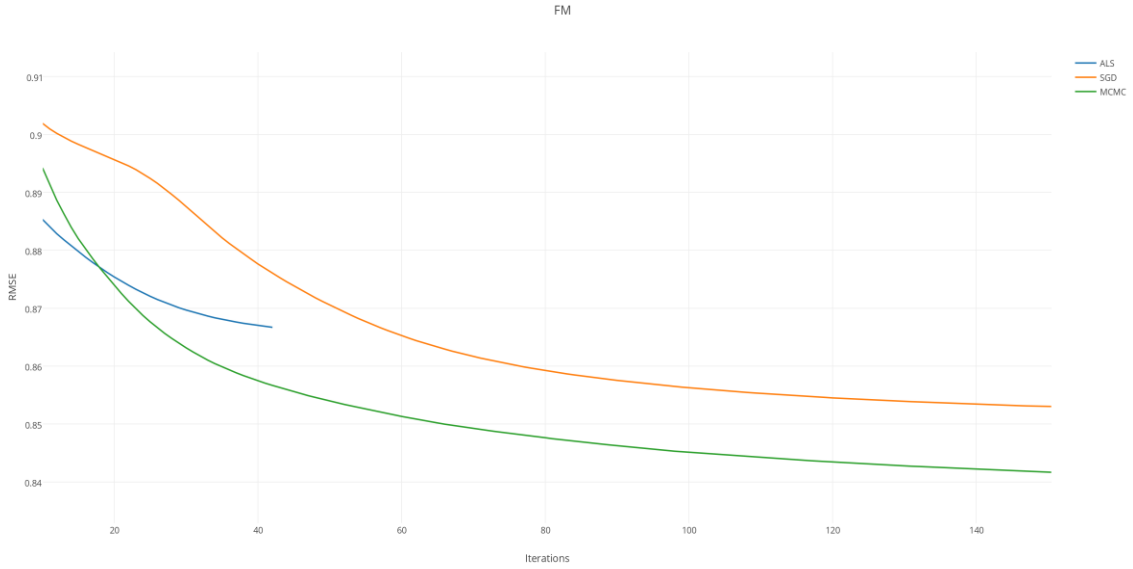

Fig 7: Comparison of different methods implemented by FM

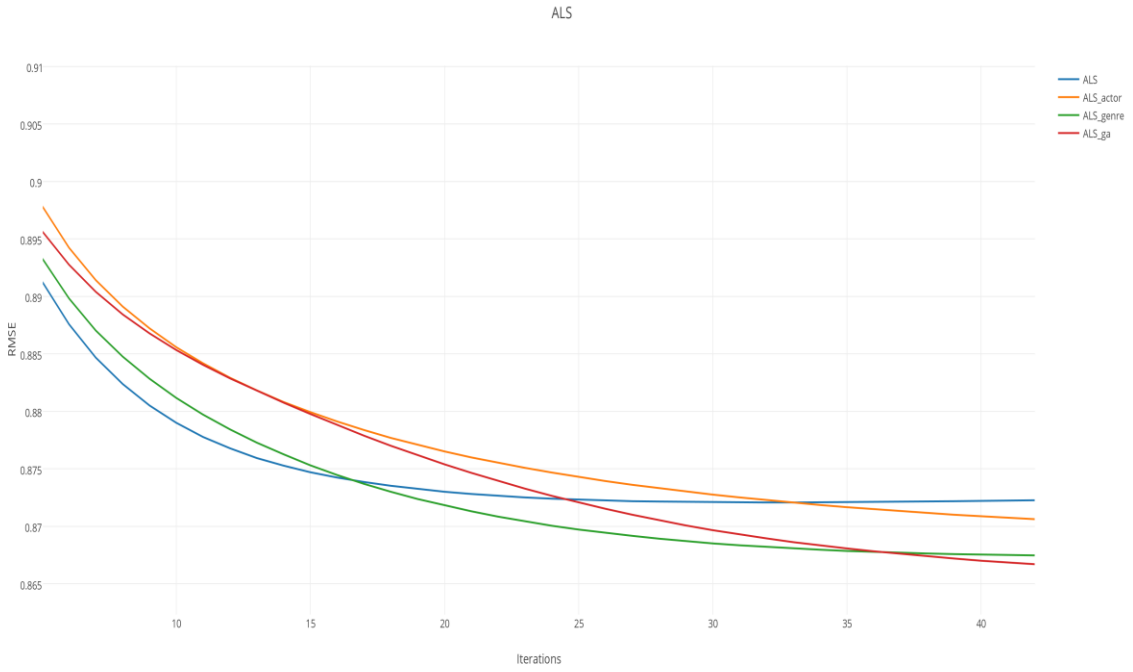

Fig 8: Different metadata added in ALS

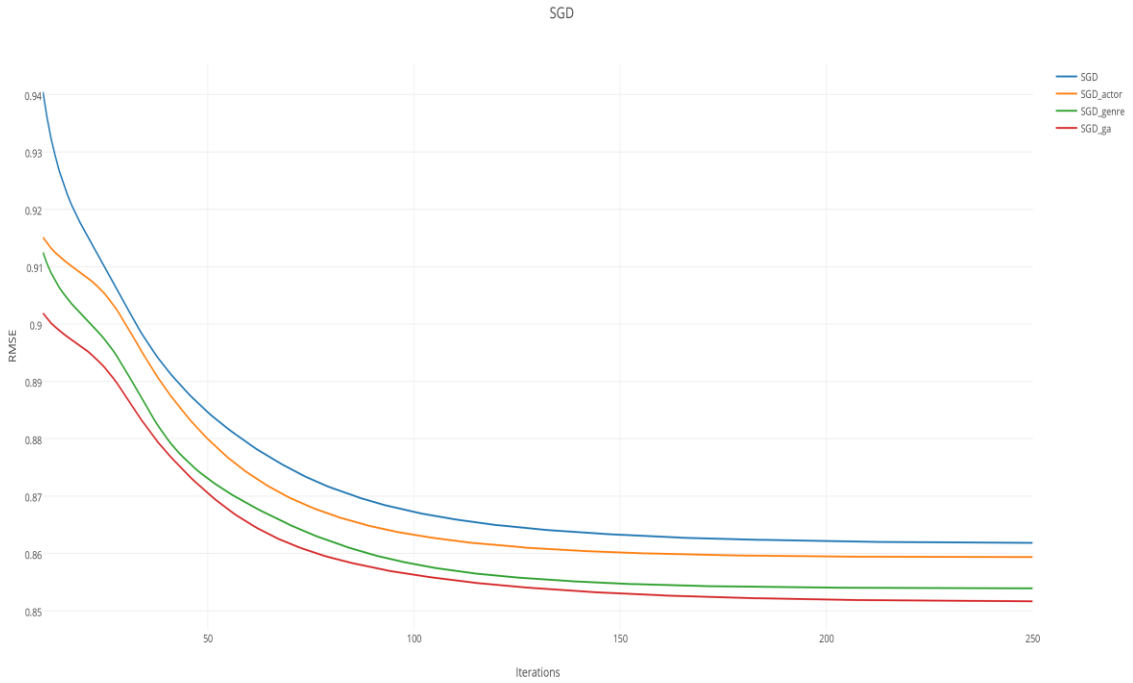

Fig 9: Different metadata added in SGD

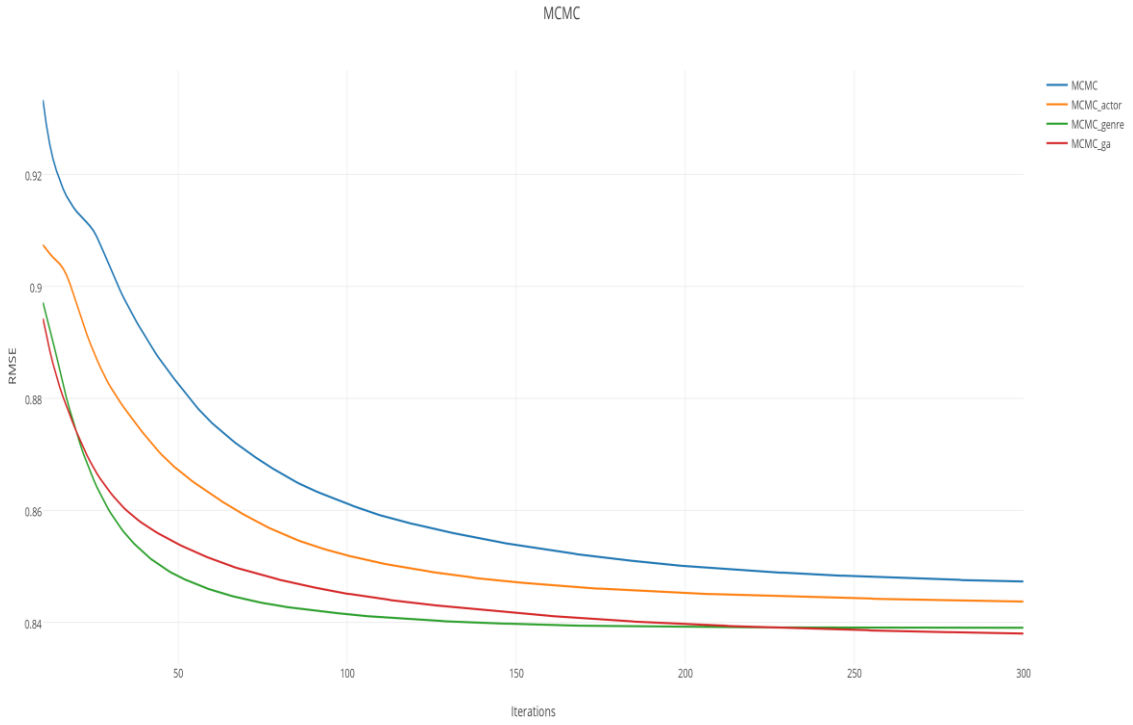

Fig 10: Different metadata added in MCMC

When I had only implemented the matrix factorization model, the following were the questions that I needed to address.

- I needed to find the reason why matrix factorization was not able to improve the results to a significant amount.
- I needed to answer whether the reason for this the algorithm itself, the poor data quality or in the way I was handling the metadata itself.
- I also needed to look for an algorithm which could work better than the matrix factorization in its base form and also could handle the information from metadata more efficiently.

Now let us address the behavior of the factorization machines that we have seen. As we already saw that the factorization handle the metadata pretty well. This is due to the fact that they are meant to handle sparse data. Our earlier model struggled to make much sense of the metadata as we were the information which was also sparse apart from the sparse base data. But in case of factorization machine which can make sense of sparse data, this was not the case. It was able to make connections when we added more sparse data.

One important thing about the factorization machine is that when we added the metadata that we got through applying the approach 1 on cast information the rmse values increased instead of decreasing. This was due to the fact that the processed data that we get after applying the approach 1, is not sparse in nature. This makes the performance of factorization machines to decrease which are not able to handle dense data. When we added the same data to the matrix factorization algorithm, the performance did not improve a much but also it did not deteriorate as well.

Now let us look how important did the preprocessing that we did with the genre and cast information prove to be. To test this we did two tests. First, instead of adding the processed metadata we tried to the data in the raw form. This means that in each row instead of adding the processed value which was between 0 and 1, we now added either a 1 or a 0. This means that if a genre or a cast is present in a movie then we put the binary value 1 else the binary value 0.

The table 8 compares the result when we added the processed knowledge about the cast of the movie to the results we got when we added information about the cast in the unprocessed form in its entirety.

As we see it performs worse than the results that we got when the processed information was added. Not only that, the results are slightly bad than that we got with no metadata information. In the second test we added random values to the factorization machine

similar to that as our metadata. The sparsity of the data was the same as that of the metadata. We observed that the rmse value was greater than that we got when no metadata was added. Both these tests indicate that the preprocessing with the data is a crucial step.

Method	**RMSE_processed**	**RMSE_unprocessed**
ALS	0.8695	0.8737
SGD	0.8604	0.8764
MCMC	0.8436	0.8470

Table 8: Rmse value for processed vs unprocessed data

Therefore now I can answer all the questions that I had after we had implemented just the matrix factorization algorithm. The matrix factorization was not able to handle the metadata as it was very sparse in nature. The data was not that bad as we thought it was initially. The preprocessing steps that I applied were working well, so I was actually handling the metadata in the right way. I found a better model than matrix factorization in MCMC. The Libfm implementation was much faster than the matrix factorization model that I had implemented ourselves. I was also able to analyze how factorization machines reacted to different type of information that I incorporated and I was able to figure out what works well with them.

I also try to add the information about the user that I had which included the age, occupation and gender of the user. I just added it to the FM as it was because there was nothing to process in this type of data. The table 9 shows the results.

Method	**RMSE**
ALS	0.8793
SGD	0.8720
MCMC	0.8441

Table 9: Rmse value after adding content about user

We see that MCMC shows a little improvement but the other algorithms show a deterioration in performance that we got from the base data.

In general MCMC apart from giving better results in terms of RMSE, is also better at making sense of data not in the best of forms as compared to the other 2 algorithms.

Chapter 7

Conclusion and Future Work

In this thesis, I tried to test how well factorization machines can handle the content metadata. It is already shown that factorization machines work well in integrating contextual metadata without much preprocessing. We concluded that it can handle the content metadata extremely well and better than the other models given the data is preprocessed.

For future work I propose two areas to look into. Firstly user side information can be processed and added to see how factorization machines use them. Secondly, as in [18], [19] I can look into techniques through which I can calculate the similar words for different tags. One such technique could be natural language processing as this would help us make more sense of the synopsis of the movie. As of now, I am just searching for the root forms of some critical words, but if I could generate other words similar to tag words, I would be able to better judge the preference of the user.

Chapter 8

References

[1] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, “GroupLens: an open architecture for collaborative filtering of netnews,” in *ACM CSCW ’94*, pp. 175–186, ACM, 1994.

[2] J. S. Breese, D. Heckerman, and C. Kadie, “Empirical analysis of predictive algorithms for collaborative filtering,” in *UAI 1998*, pp. 43–52, AAAI, 1998.

[3] J. Herlocker, J. A. Konstan, and J. Riedl, “An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms,” *Information Retrieval*, vol. 5, no. 4, pp. 287–310, 2002.

[4] Michael J. Pazzani, Daniel Billsus: Content-Based Recommendation Systems. The Adaptive Web 2007: 325-341

[5] Töscher, A., Jahrer, M., & Bell, R. M. (2009). The bigchaos solution to the netflix grand prize. Netflix 322 prize documentation

[6] Paterek, A. (2007, August). Improving regularized singular value decomposition for collaborative filtering. 324 In Proceedings of KDD cup and workshop (Vol. 2007, pp. 5-8).

[7] James Bergstra, Yoshua Bengio; Random Search for Hyper-parameter Tuning, 13(Feb):281−305, 2012

[8] C. Volinsky et al.: “Matrix Factorization Techniques for Recommender Systems” In: IEEE Computer, Vol. 42 (2009) , pp. 30-37 .

[9] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Reidl, “Item-based collaborative filtering recommendation algorithms,” in *ACM WWW ’01*, pp. 285–295, ACM, 2001

[10] G. Karypis, “Evaluation of item-based top-N recommendation algorithms,” in *ACM CIKM ’01*, pp. 247–254, ACM, 2001.

[11] Ajit P. Singh, Geoffrey J. Gordon, “Relational Learning via Collective Matrix Factorization”, CMU-ML-08-109, June 2008

[12] Stephen A. Vavasis, “On the complexity of non-negative matrix factorization”, *SIAM J. Optim.*, 20(3), 1364–1377, 2007.

[13] Steffen Rendle (2010): Factorization Machines, in Proceedings of the 10th IEEE International Conference on Data Mining (ICDM 2010), Sydney, Australia.

[14] Steffen Rendle (2012): *Factorization Machines with libFM*, in ACM Trans. Intell. Syst. Technol., 3(3), May.

[15] Steffen Rendle, Zeno Gantner, Christoph Freudenthaler, and Lars Schmidt-Thieme. Fast context-aware recommendations with factorization machines. In Proceedings of the 34th ACM SIGIR Conference on Research and Development in Information Retrieval. ACM, 2011.

[16] Christoph Freudenthaler, Lars Schmidt-Thieme, Steffen Rendle (2011): *Bayesian Factorization Machines*, in Workshop on Sparse Representation and Low-rank Approximation, Neural Information Processing Systems (NIPS-WS), Granada, Spain.

[17] Steffen Rendle (2012): *Learning Recommender Systems with Adaptive Regularization*, in Proceedings of the 5th ACM International Conference on Web Search and Data Mining (WSDM 2012), Seattle.

[18] T. Joachims, “Optimizing search engines using clickthrough data,” in *KDD ’02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2002, pp. 133–142.

[19] I. Pil´aszy and D. Tikk. Recommending New Movies: Even a Few Ratings Are More Valuable Than Metadata. In RecSys ’09: Proceedings of the Third ACM Conference on Recommender Systems, pages 93–100. ACM, 2009.

[20] Gediminas Adomavicius, Alexander Tuzhilin: Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. IEEE Trans. Knowl. Data Eng. 17(6): 734-749 (2005)

[21] Peter Forbes, Mu Zhu: Content-boosted matrix factorization for recommender systems: experiments with recipe recommendation. RecSys 2011: 261-264

[22] Oleksandr Krasnoshchok, Yngve Lamo: Extended Content-boosted Matrix Factorization Algorithm for Recommender Systems. KES 2014: 417-426

[23] Yehuda Koren: Collaborative filtering with temporal dynamics. Commun. ACM 53(4): 89-97 (2010)

[24] Li Chen, Guanliang Chen, Feng Wang: Recommender systems based on user reviews: the state of the art. User Model. User-Adapt. Interact. 25(2): 99-154 (2015)

[25] Alexandros Karatzoglou, Xavier Amatriain, Linas Baltrunas, Nuria Oliver: Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. RecSys 2010: 79-86

[26] Hendrik Wermser, Achim Rettinger, Volker Tresp: Modeling and Learning Context-Aware Recommendation Scenarios Using Tensor Decomposition. ASONAM 2011: 137-144

[27] Ruslan Salakhutdinov, Andriy Mnih: Bayesian probabilistic matrix factorization using Markov chain Monte Carlo. ICML 2008: 880-887

[28] Weike Pan, Zhuode Liu, Zhong Ming, Hao Zhong, Xin Wang, Congfu Xu: Compressed knowledge transfer via factorization machine for heterogeneous collaborative recommendation. Knowl.-Based Syst. 85: 234-244 (2015).

[29] Weike Pan, Shanchuan Xia, Zhuode Liu, Xiaogang Peng, Zhong Ming: Mixed factorization for collaborative recommendation with heterogeneous explicit feedbacks. Inf. Sci. 332: 84-93 (2016)

[30] Babak Loni, Martha Larson, Alexandros Karatzoglou, Alan Hanialic: Recommendation with the Right Slice: Speeding Up Collaborative Filtering with Factorization Machines. RecSys Posters 2015