ABSTRACT

Latent Factor Models have been extensively used to predict rating in many recommender systems. There are different types of Latent Factor Models, such as MF, BSVD, SVD++, etc. The key idea behind Latent Factor Models is that ratings on items are influenced by a set of factors (e.g., the number of actions, combination of genres, etc.), which are in general not obvious but can be captured through mining the rating history. Most recommendation algorithms based on latent factor models then focus heavily upon the traditional user-item rating data. However, user ratings are only one of the many data sources we can get. In real world, we can acquire abundant auxiliary information that is associated with items. This information might determine the user's intent to purchase or rating the item. For instance, in the movie recommender systems, there is information about the release date, genres and director of a film, all of which might influence the users' intent on rating the movie. A new release movie is usually chosen by customers despite they may lack interest in the movie as predicted by the Latent Factor Models.

This project proposed a novel approach to integrating item attributes with Latent Factor Models. This approach takes into consideration, not only the interactions between users and items, but also the interactions between users and item attributes that will influence the users' preferences on items. The experimental results show that our approach can improve the accuracy of rating prediction for both the basic Latent Factor Model (Matrix Factorization) and the state-of-the-art model advocated by Koren and Bell (BSVD).

Keywords: recommender system, collaborative filtering, latent factor model

1 Introduction

We are living in the age of personalized recommendations. With the help of recommendation engines, entrepreneurs can target products to specific consumer groups, consumers can find what they are really interested in from tens of thousands of products. We use recommendation engines every day, intentionally and otherwise. Many larger e-commerce websites have integrated recommendation tools to their online systems in order to enhance user experiences and create new revenue. Examples of these websites include book recommendations at Amazon [4], news at Google [5], and Movies at Netflix [6].

Generally speaking, a recommendation problem is to predict the rating that a user would give to an item [7,8], or predict if a user would rate an item based on his/her personal preference. The input (also known as training data or observed rating data) of the problem is a sparse two-dimensional rating matrix, in which one dimension represents users and the other represents items. The matrix is sparse because a typical user only rates a tiny fraction of all available items. The magic of recommendation engines is that it fills in the missing entries of the sparse matrix by analyzing the observed rating data.

Recommendation algorithms can be divided into two broad categories: Content-based Recommendation and Collaborative Filtering (CF). Content-based Recommendation provides the recommendation by analyzing item descriptions and user profiles. By contrast, CF provides personalized recommendations by taking advantage of historical data of user preferences, based on the assumption that people will like similar kinds of items as they liked in the past or items that are highly rated by those who share similar interests. One of the advantages of CF is that it is capable of recommending complex items such as movies without requiring an "understanding" of the item itself.

There are two main CF approaches: neighborhood methods and latent factor models. Neighborhood methods, also known as kNN, use similarity functions such as the Pearson Correlation or Cosine Distance to cluster similar users and items. The recommendations are then computed by using data from those neighbors. On the other hand, latent factor models overcome the prediction challenge by decomposing the training matrix into two low-rank matrices, and using the dot products of the two matrices to approximate the real rating matrix. Many research works show that latent factor models can produce more accurate predictions than neighborhood-based methods [1][2]. Since the Netflix Prize competition [26], latent factor models have been applied extensively for the implementation of recommender systems, especially in rating prediction. The most commonly used metric for comparing recommendation engines is the root mean squared error (RMSE), and minimizing RMSE by devising new adaptations has become a trend for accurate recommendation [9].

Although the latent factor models have been applied successfully in many recommender systems, to my best knowledge, most recommendation models based on latent factor models do not make proper use of some annotated attributes that are associated with items, such as release date, genre, director in the movie recommender systems. In fact, the rating on a specific movie always reflects a user's preference to the corresponding attributes of the movie. For instance, some people tend to give a higher rating to an old movie because they are aged and nostalgic, the release date attribute decides their attitudes toward the old movie. Although the user's preferences on these informative attributes cannot be observed explicitly, we can infer their interesting patterns by exploiting the incremental rating behaviors.

This project aims at building a predictor by accumulating user preferences on each attribute of items, we call these attributes Impact Factors. Our model extends the latent factor models, and calculates the interactions between users and items, as well as between users and Impact Factors in the same latent space, then integrates the weights of these interactions to get the final rating decisions. The experimental results show that prediction accuracy can be improved by our new extension.

The main contributions of this project include:

i. In addition to the user-item ratings, a new model is proposed to leverage the explicit item attributes. The experimental results show that the new model achieves better performance than the latent factor models that do not make use of item attributes.

ii. Our approach is lightweight and highly extensible, it can be easily applied to other latent factor model based recommendation algorithms.

2 Related work

In this section, we briefly present some of the research literature related to CF and the combination of content-based and latent factor based recommendation models.

Amr Ahmed *et al.* [22] proposed a hybrid model LFUM, which combines the user preferences with the latent factor models. LFUM learns user preferences over item by a personalized Bayesian hierarchical model, then integrates these learned preferences with a latent factor model and trains the resulting system end-end to optimize a Bayesian discriminative ranking loss function. Thus, the hybrid model can overcome the cold start problem and provide the advantages of both of the latent factor models and content-based models.

Although most of the work on CF has focused on the traditional two-dimensional user-item problem, Alexandros Karatzoglou *et al.* [23] introduced a CF method based upon Tensor Factorization, called Multiverse Recommendation. Instead of the traditional 2D User-Item matrix,

Multiverse Recommendation models the data as a User-Item-Context 3D tensor shown in Fig.1. This decomposition model allows for full control over the dimensionality of the factors extracted for the users, items and context by adjusting configuration parameters. In addition, different types of context can be considered as additional dimensions in the representation of the data as a tensor. The factorization of this tensor leads to a compact model of the data which can be used to provide context-aware recommendations.

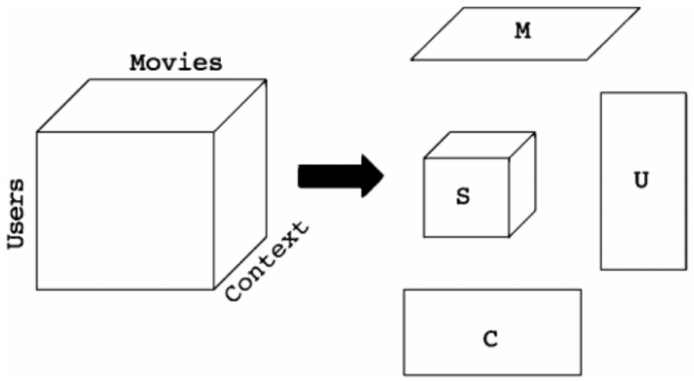

Figure 1: Illustration of the 3-dimensional tensor factorization model[23].

Other papers also introduced time effect into latent factor models. Xiong *et al.* [24] present a Bayesian Probabilistic TF model to capture the temporal evolution of online shopping preferences. Koren *et al.* [25] also introduced a time effect into the factor model called TimeSVD++ which significantly improves the Root Mean Squared Error (RMSE) on the Netflix data compared to previous non-temporal factorization models.

However, compared with other variants of latent factor models, our approach is straightforward and readily comprehensible, which is the main contribution of our work and will be presented next in detail.

3 Preliminaries

3.1 Movie Recommender Systems

Movie Recommender Systems help millions of people narrow the universe of potential films to fit their unique tastes. These services depend on the synthesizing application of machine-learning strategies, which breaks down the movies into long lists of attributes and matches these elements to a user's preferences.

Netflix, an on-demand Internet streaming media provider, is the earliest pioneer in Movie Recommender Systems. Statistics shows that 75% of what people watch at Netflix is from the recommendations generated from the adapted recommendation algorithms [26]. The core service

that Netflix provides is ranking (also known as Top-N recommendation), deciding what order to place the movies in a rack. The goal of the ranking service is to find the best possible ordering of a set of movies for a member based on his/her taste preferences, within a specific context, in real-time [21]. Netflix decomposes ranking into scoring, sorting, and filtering sets of movies for presentation to a user.

The prediction model used by Netflix includes two features: Predicted Rating and Popularity. The model combines the two features linearly by using the equation [21]

$$f_{rank}(u, v) = w_1\, p(v) + w_2\, r(u, v) + b,$$

where u denotes user, v denotes video item, p denotes popularity and r denotes predicted rating, w_1 and w_2 denote the weight for each feature. The bias b is a constant and will not affect the final ordering. This equation defines a two-dimensional space shown in Fig. 2. Setting the weights is a machine learning problem: select positive and negative examples from historical data and let a learning algorithm learn the weights that optimize the model.

Figure 2: The two-dimensional prediction model of Netflix[21]

3.2 Notation

This project uses many notations. Table 1 lists some main notations all through the report. Users are denoted by u, and items are denoted by i. There are three types of Impact Factors in our work: release date, genre and director. We let r(i) and gen(i) denote the release date and genres of item i respectively, we also use d(i) to denote the director of item i.

Symbols	Definition
$R = \{r_{ui}\}_{n_u x n_i}$	the user-item rating matrix, where each element r_{ui} represents the observed rating of user u towards item i. n_u denotes the number of users, and n_i denotes the number of items
$R_0 = \{0\}_{n_u x n_i}$	the user-item rating matrix where the rating is unknown, namely, $r_{ui} = 0$
r'_{ui}	the predicted rating for user u and item i
N	the total number of observed ratings
N(x)	the total number of observed ratings on attribute x
f	the number of latent features for users and items
b_u	bias for user u. It is a real number for each user that captures how lenient or critical user u is
b_i	bias for item i. A real number for each item that shows how liked item i is. For example, b_i is expected to be a negative value if item i is a terrible movie
P	the user latent factor matrix, $P = [p_u]$, where $p_u \in \mathbb{R}^f$, $u = 1...n_u$, is the u^{th} column of P
p_u	the vector of latent factor values for user u. It is used to characterize user u by f real values
Q	the item latent factor matrix, $Q = [q_i]$, where $q_i \in \mathbb{R}^f$, $i = 1... n_i$, is the i^{th} column of Q
q_i	the vector of latent factor values for item i. It is used to characterize item i by f real values
e_{ui}	the absolute error between the predicted rating r'_{ui} and the real rating r_{ui}
λ	the regularization parameter
γ	the learning rate
L_{base}	the right-hand side of the standard latent factor model equations. For MF, $L_{base} = p_u q_i^T$. For BSVD, $L_{base} = \mu + b_u + b_i + p_u . q_i^T$

Table 1: Notations

3.3 Recommendation Strategies

There are two main strategies that can be applied in recommender systems.

3.3.1 Content-based Filtering

Content-based recommendation algorithms analyze item descriptions to identify items that are of particular interests to a user [10]. The algorithms try to recommend items that are similar to the items that a user liked in the past by comparing their features. The difficulty of implementing content-based recommender systems is that the algorithm needs to learn user preferences from single content source (e.g., the news browsing history) and to use them across other content types (e.g., music, videos, products, discussions, etc.).

3.3.2 Collaborative Filtering

Instead of user profiles and item descriptions, CF algorithms take advantage of history data (e.g., the rating history) [23], which makes the approach independent from the domain. For example, in a typical movie recommender system, a user first provides ratings of some movies, then the system recommends other movies based on ratings already provided by other users who share similar rating patterns with him. CF focuses on three important factors: users interest patterns, item properties and other factors that directly affect users' preferences over items. In practice, various kinds of information can be used to model these factors. For instance, users' browsing history over movie reviews may correlate with users' taste over movies; the information of the director and actors of a movie can be used to predict its properties; the rating history over similar movies directly affects whether a user will favor the current one.

3.4 Latent Factor Models

Latent factor models are commonly applied in CF. The models assume that recommendation can be inferred from high correspondence between item and user factors. A typical latent factor model maps users and items to a joint latent factor space of dimension *f*, such that users' rating processes on items are modeled as dot products in the space. Matrix Factorization(MF) is one of the most successful implementations of latent factor models [3]. Given a rating matrix R, the purpose of Matrix Factorization is to find two low-rank latent factor matrices P(|U|×*f*) and Q(|I|×*f*), which can minimize the following Frobenius norm

$$||R - P*Q^T||_{F,}$$

which can also be equally modeled by

$$R \approx P*Q^T$$

That means finding two low-rank matrices to approximate the user-item rating matrix R. Traditional ways to lower matrix dimension are using SVD [11] or PCA [12] on matrix R. However, applying them directly on a sparse matrix may raise many issues. First, imputation needs

to be done to fill in the missing ratings when preprocessing, which may be very expensive as it significantly increases the amount of data. In addition, inaccurate imputation may lead to data considerably distorted. Hence, more recent research suggests modeling directly on the observed ratings, meanwhile avoiding over-fitting through a regularized model. A commonly used learning algorithm to train P and Q is the Stochastic Gradient Descent. I will elaborate it in the next section.

As a result, each user is associated with a vector of latent factor values p_u, and each item is associated with a vector of latent factor values q_i, these latent factors are in general not obvious but can be inferred from explicit user feedback. In the movie recommender systems, these factors might measure the number of actors, portion of actions, or director's popularity. If a user loves watching action movies by popular directors, the p_u would have relatively larger values on the two factors. If a movie also has large values on the two factors, this movie will be recommended to the user because the dot product of p_u and q_i would be relatively large. Once we get P and Q, the predicted rating can be represented by

$$r'_{ui} = p_u q_i^T$$

Koren and Bell [29] advocated an extended model based on the standard latent factor model. In this model, the prediction of user u's rating on item i is made by:

$$r'_{ui} = \mu + b_u + b_i + p_u . q_i^T,$$

where μ is a global average rating of all knowing ratings, b_u is user bias, which measures the user's tendency to rate the items, b_i is item bias, which measures the tendency of the item's ratings in the data sets. For simplicity, consider the recommended items are movies, the bias of a movie would describe how well this movie is rated compared to the average, across all movies. This depends only on the movie and does not take into account the interaction between a user and the movie. Similarly, a user's bias corresponds to that user's tendency to give better or worse ratings than the average. Finally, the interaction term $p_u . q_i^T$ describes the interaction between the user and the movie, i.e., the user's preference for that movie. In general, b_u and b_i are initially set to 0, and p_u and q_i are assigned to random numbers that are proportional to the square root of *f*. Through the learning algorithm intercepted in the next section, these parameters will be updated to optimal values. For instance, we want to predict user Frank's rating on movie Minions. Suppose the average rating over all movies, μ, is 3.8. Furthermore, the learning algorithm discovers the quality of Minions is 0.5 stars higher than the average movies, say, $b_i = 0.5$. On the other hand, Frank is a critical user and he tends to rate 0.2 stars lower than the average, $b_u = -0.2$. Moreover, Frank's preference on Minions, $p_u . q_i^T$, is 0.5. Thus, the rating prediction for user Frank on Minions will be $3.8 + 0.5 - 0.2 + 0.5 = 4.6$.

3.5 Learning algorithm

Both SGD and ALS can be used to train P and Q for latent factor models.

3.5.1 Stochastic Gradient Descent (SGD)

As previously mentioned, we need an effective algorithm to train P and Q, one commonly used algorithm is the full gradient descent [13], where in each iteration the full gradient over all training data is computed and used to update the data. However, this is found to have slow convergence for training latent factor models over large data sets. This project uses another type of algorithm, which is called Stochastic Gradient Descent.

Stochastic Gradient Descent (SGD) tries to minimize the squared error

$$\sum_{(u,i)\in I} \left(r_{ui} - p_u q_i^T\right)^2$$

for each existing r_{ui} entry. Taking a standard latent factor model for example, the purpose of the model is to factorize the user-item rating matrix in the following way:

$$R \approx P * Q^T$$

then utilize the factorized user-specific and item-specific matrices to make further missing data prediction.

$$r'_{ui} = p_u q_i^T$$

Traditional way to approximate a rating matrix R is to minimize

$$\left\|R - PQ^T\right\|^2 \quad (1)$$

However, in real world, each user only rates a very small portion of items, the matrix R is usually extremely sparse, and we only need to factorize the observed ratings in matrix R. Hence, we change Eq. (1) to

$$\min_{P,Q} \sum\nolimits_{(u,i)\in I} \left(r_{ui} - p_u q_i^T\right)^2, \quad (2)$$

In order to avoid overfitting, two regularization terms are added into Eq. (2), then we have

$$\min_{P,Q} \sum\nolimits_{(u,i)\in I} \left(r_{ui} - p_u q_i^T\right)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2), \quad (3)$$

where λ is the weight parameter for the regularized term. Eq. (3) is a cost function with local minimum. To compute the factor vectors (p_u and q_i), the approach loops through all ratings in the training set R. For each given training case, the approach predicts r_{ui} and computes the associated prediction error.

$$e_{ui} = r_{ui} - p_u q_i^T \quad (4)$$

Then P and Q can be updated by moving in the opposite direction of the gradient for each training case.

$$\frac{\partial J}{\partial p_u} = -e_{ui}.q_i + \lambda p_u$$

$$\frac{\partial J}{\partial q_i} = -e_{ui}.\, p_u + \lambda q_i$$

$$p_u \leftarrow p_u + \gamma(e_{ui} q_i - \lambda p_u) \tag{5}$$

$$q_i \leftarrow q_i + \gamma(e_{ui} p_u - \lambda q_i)$$

where J denotes Eq. (3). In practice, vectors of latent factor values p_u and q_i are initially assigned to small random numbers. γ decides the convergence rate for the cost function. In most cases, γ is regarded as an empirical parameter, adapting to different data sets. The regularization parameter λ allows us to trade off between model accuracy and complexity.

3.5.2 Alternating Least Squares (ALS)

ALS represents a different approach to optimizing the loss function. Although SGD can be implemented easily and has a relatively fast running time, it is beneficial to use ALS when handling with large scale datasets[14]. To solve Eq. (3), ALS fixes one of the unknowns (p_u or q_i) and changes the non-convex optimization problem into an easy quadratic problem. One of the benefits of ALS is that p_u and q_i can be updated in parallel under distributed computing environments. This project uses SGD due to its ease implementation.

3.6 Datasets

Our model will be trained and tested on the MovieLens 100K data which contains 100K ratings on a scale of 1 to 5 for 1682 movies and 943 users. All the data covered from September 19th, 1997 through April 22nd, 1998. MovieLens datasets have been widely used in many CF experiments [15, 16, 17] along with Netflix dataset. Its sparsity is 95.53% and the minimum number of ratings given by users is 20, which is a quite denser compared with Netflix dataset and Epinions dataset [18]. 84% ratings are greater than or equal to 3, this means most of the users tend to rate these movies they enjoy. MovieLens datasets not only provide ratings of different movies given by different users but also provide the timestamps when these ratings were assigned, as well as auxiliary information about the movies such as release date, genres, director, etc. Hence, this dataset can help us evaluate the new model effectively.

4 Integration of Item Attributes

The latent factor models do not take item attributes into consideration. In reality, we can usually collect the additional information about the items. Fig.3 shows all relationships among the data entities in the MovieLens. Our work combines some of these attributes (director, release date and genres) with latent factor models to improve recommendation accuracy.

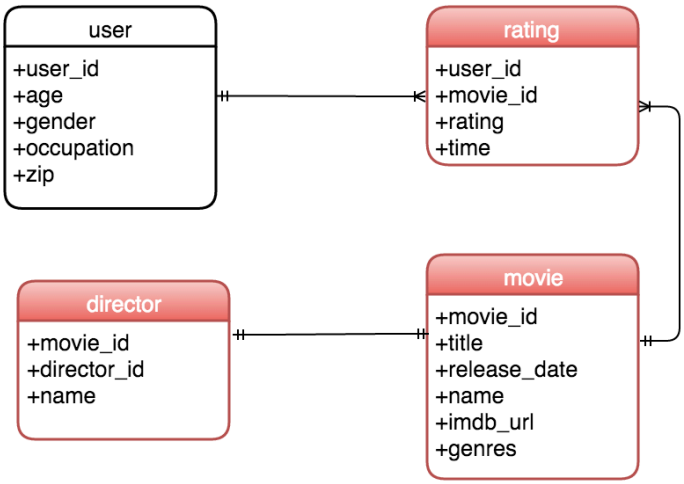

Figure 3: Entity-Relationship Diagram of MovieLens Dataset

4.1 Release Date

The release date of a film plays a significant role on user's rating decision which is mainly represented in the following two aspects.

4.1.1 Popularity

Generally, the movies will be popular when they just came out and lose popularity when time goes on. Fig. 4 demonstrates the relationship between average rating and time of release. From Fig. 4 (a) we know that for most movies, their average ratings change dramatically in the first few years, but after a period of time, the average ratings become stable. That means, the users' attitudes toward a movie will be fixed after a period of time. In addition, we choose five movies randomly from MovieLens 10M datasets and plot their average ratings with years after their release. We found people tend to give strict ratings as the release date grows, which is shown in Fig. 4 (b).

It is not difficult to capture the time effect in latent factor models by using a time-dependent item bias model:

$$r'_{ui} = L_{base} + v_i z_t$$

L_{base} stands for the right-hand side of the standard latent factor model equations. For MF, L_{base} is $p_u q_i^T$. For BSVD L_{base} is $\mu + b_u + b_i + p_u . q_i^T$. t is the number of days after movie i was released. Both v_i and z_t are two latent factor vectors for *i* and *t*. the dot products of them represent the fluctuation of item i's popularity with the number of days *t*.

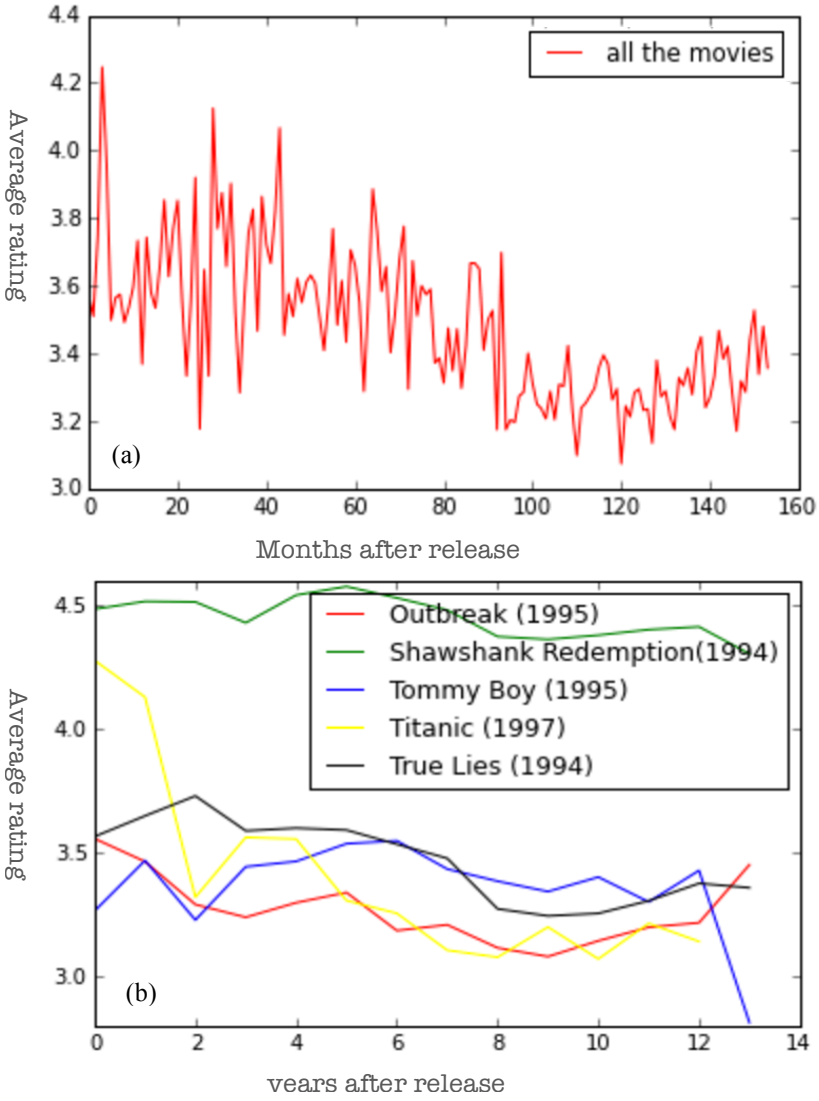

Figure 4: Rating distribution with time of release

4.1.2 Movie Elements

The interest and habit of the whole society change over time. At different ages, people enjoy different things. Since the movies always try to cater to contemporary's tastes, the elements in movies also vary with time. These elements are not restricted to the nineteen genres listed in MovieLens datasets, they are usually not obvious, and difficult to summarize. For example, some types of movies, such as the movies that dominate the summer and Christmas holidays, are very popular in youngsters due to the special effects and novel storylines. On the other hand, old movies always attract aged people due to some classic elements such as old-fashioned romantic comedy. Hence, users have a wide range of interest patterns associated with release date. Fig. 5 demonstrates the rating behavior for people of different ages. We can find that young people (under

25) prefer to giving five-star ratings to recent released movies, while aged people (over 50) tend to give five-star ratings to relatively old movies.

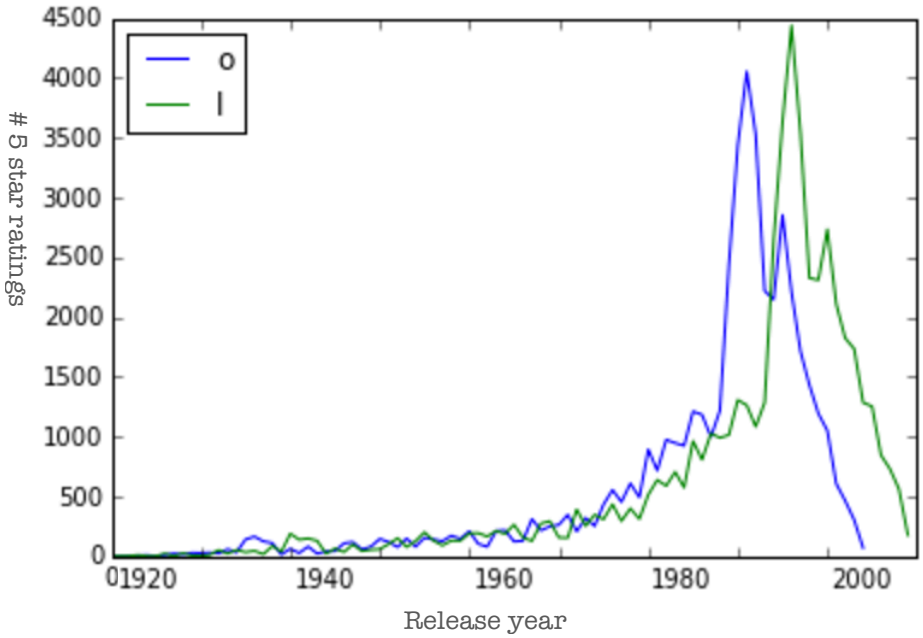

Figure 5: Difference of rating behavior among different population

To model the influence that release date brings to user, we augment the user bias b_u with additional biases for release date, which adjusts the predicted rating based on user's preference over the release date.

$$r'_{ui} = L_{base} + \frac{1}{\sqrt{|N(r(i))|}} p_u q^T_{r(i)} \tag{6}$$

A user who showed a preference for release date of item i is characterized by $p_u q^T_{r(i)}$, we use $\frac{1}{\sqrt{|N(r(i))|}}$ to normalize the preference, $|N(r(i))|$ is the total number of observed ratings on movies released the same day as movie i.

4.2 Director

The director information provided by MovieLens is also useful in prediction problem. Instinctively, a user is likely to rate higher the movies that were directed by his/her favorite directors. Meanwhile, a prestigious director's works probably get higher ratings even if their qualities are not outstanding. Similar with release date, we can also model the influence that the director factor brings to the user

$$r'_{ui} = L_{base} + \frac{1}{\sqrt{|N(d(i))|}} p_u q^T_{d(i)}, \tag{7}$$

where $|N(d(i))|$ is the total number of observed ratings on movies directed by the same director

as movie i.

4.3 Genres

We also considered genres data in the first place. However, the experimental results show that the genres data have little impact on final results. The reasons for this may include: First, each movie has many different genre labels. We tried many ways to group and weight those labels, unfortunately which does not work very well. Second, there are some human factors in labeling the genres data, in other words, the genres data itself is inaccurate to some extent.

5 Implementation

Based on Eqs. (5)(6)(7), the impact factors that influence user's preference on item can be generalized as following:

$$r'_{ui} = L_{base} + v_i z_t + \sum_{j \in S} \frac{1}{\sqrt{|Nr_j|}} x_{uj} y_j^T, \tag{8}$$

where S stands for the set of impact factors. In the movie recommender systems, impact factor set includes release date, genres and director, etc. $x_{uj} y_j^T$ denotes the interaction between user u and impact factor j. $\frac{1}{\sqrt{|Nr_j|}}$ is the normalization parameter. Fig. 6 shows an example how a user weights impact factors into the last rating decision. In this example, release date and director are taken as the underlying factors that users might consider when giving rate to the movie. Firstly, we measure user u6's interest in movie Avatar by computing the dot product of corresponding user-item latent vectors. Next, we know the release date of Avatar is 12/2009, so u6's preference on 12/2009 is computed. Finally, we compute u6's preference on the director, James Cameron. The final prediction of rating on Avatar is made by a simple linear combination of u6's preference on each impact factor.

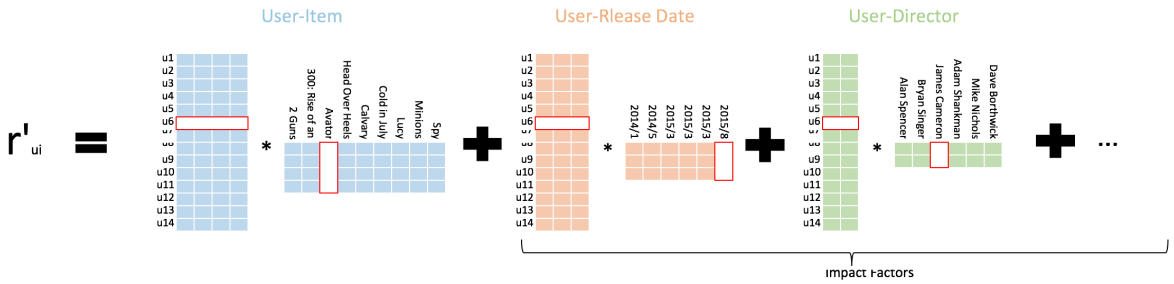

Figure 6: User's rating procedure in movie recommender systems

Eq. (8) demonstrates how to compute user's preference over the item when taking impact factors into consideration. We need to train those feature specific matrices from the given training data. Following Eq. (3), the cost function with impact factors can be modified as:

$$\sum_{(u,i)\in I} (r_{ui} - \mathrm{r'_{ui}})^2 + \lambda \left(\|p_u\|^2 + \|q_i\|^2 + \|\mathrm{v_i}\|^2 + \|z_t\|^2 + \sum_{j\in S} \left\|x_{uj}\right\|^2 + \sum_{j\in S} \left|Nr_j\right|^{-1} \left\|y_j\right\|^2 \right) \tag{9}$$

With Eq. (5), the gradients of the involved parameters can be calculated by following rules:

$$\frac{\partial \mathrm{J}}{\partial \mathrm{p_u}} = -\mathrm{e_{ui}}.\,\mathrm{q_i} + \lambda \mathrm{p_u}$$

$$\frac{\partial \mathrm{J}}{\partial \mathrm{q_i}} = -\mathrm{e_{ui}}.\,\mathrm{p_u} + \lambda \mathrm{q_i}$$

$$\frac{\partial \mathrm{J}}{\partial \mathrm{v_i}} = -\mathrm{e_{ui}}.\,\mathrm{z_t} + \lambda \mathrm{v_i}$$

$$\frac{\partial \mathrm{J}}{\partial z_{\mathrm{t}}} = -\mathrm{e_{ui}}.\,v_i + \lambda z_{\mathrm{t}}$$

$$\frac{\partial \mathrm{J}}{\partial x_{uj}} = -\frac{1}{|Nr_j|}.\,\mathrm{e_{ui}}.\,y_j + \lambda x_{uj}$$

$$\frac{\partial \mathrm{J}}{\partial \mathrm{y_j}} = -\mathrm{e_{ui}}.\,x_{uj} + \frac{\lambda}{|Nr_j|} y_i$$

Where J denotes Eq. (9). Similar with the SGD rules for updating $\mathrm{p_u}$ and $\mathrm{q_i}$ at each iteration, the updating rule for model parameters can be formulated by:

$$\mathrm{p_u} \leftarrow \mathrm{p_u} + \gamma(\mathrm{e_{ui}q_i} - \lambda \mathrm{p_u})$$

$$\mathrm{q_i} \leftarrow \mathrm{q_i} + \gamma(\mathrm{e_{ui}p_u} - \lambda \mathrm{q_i})$$

$$\mathrm{v_i} \leftarrow \mathrm{v_i} + \gamma(\mathrm{e_{ui}z_t} - \lambda \mathrm{v_i})$$

$$\mathrm{z_t} \leftarrow \mathrm{z_t} + \gamma(\mathrm{e_{ui}v_i} - \lambda \mathrm{z_t})$$

$$\mathrm{x_{uj}} \leftarrow \mathrm{x_{uj}} + \gamma(\frac{1}{|\mathrm{Nr_j}|}\mathrm{e_{ui}y_i} - \lambda \mathrm{x_{u_j}})$$

$$\mathrm{y_j} \leftarrow \mathrm{y_j} + \gamma(\mathrm{e_{ui}x_{uj}} - \frac{\lambda}{|\mathrm{Nr_j}|}\mathrm{y_i})$$

A sample implementation for the new latent factor model is illustrated in the following:

Algorithm 1: Train our new model (Impact Factors: Time effect, Release Date and Director)

Input: observed rating set R, latent factor number *f*, iteration times #Iter,
Item-Release Map IR, Item-Director Map ID
Output: trained parameters

Randomly initialize parameters of the new model
Repeat
  For each rated user-item pair in R
    Get release date r(i) and director d(i) from IR and ID
    Compute r'_{ui} with Eq.(8)
    $e_{ui} = r_{ui} - r'_{ui}$
    $t = t_{ui} - r(i)$
    $b_u = b_u - \gamma(e_{ui} - \lambda b_u)$
    $b_i = b_i - \gamma(e_{ui} - \lambda b_i)$
    For k=0; k<f; ++k
      $p_{u,k} \leftarrow p_{u,k} + \gamma(e_{ui} q_{i,k} - \lambda p_{u,k})$
      $q_{i,k} \leftarrow q_{i,k} + \gamma(e_{ui} p_{u,k} - \lambda q_{i,k})$
      $v_{i,k} \leftarrow v_{i,k} + \gamma(e_{ui} z_{t,k} - \lambda v_{i,k})$
      $z_{t,k} \leftarrow z_{t,k} + \gamma(e_{ui} v_{i,k} - \lambda z_{t,k})$
      $p'_{u,k} \leftarrow p'_{u,k} + \gamma(\frac{1}{|N(r(i))|} e_{ui} q'_{r(i),k} - \lambda p'_{u,k})$
      $q'_{r(i),k} \leftarrow q'_{r(i),k} + \gamma(e_{ui} p'_{u,k} - \frac{\lambda}{|N(r(i))|} q'_{r(i),k})$
      $p''_{u,k} \leftarrow p''_{u,k} + \gamma(\frac{1}{|N(d(i))|} e_{ui} q''_{d(i),k} - \lambda p''_{u,k})$
      $q''_{d(i),k} \leftarrow q''_{d(i),k} + \gamma(e_{ui} p''_{u,k} - \frac{\lambda}{|N(d(i))|} q''_{d(i),k})$
    End For
  End For
  Calculate the RMSE on R
Until RMSE does not improve (or iteration > #Iter)

6 Evaluation

This section provides the experimental evaluation of our approach. First, we implement two baseline models using python: Matrix Factorization and BSVD. We then extend them with Eq. (8). Finally, we analyze the improvement of our approach by comparing its recommendation result against the two baseline models.

6.1 Evaluation Metrics

The metrics for performance of recommendation models can be classified into two classes:

prediction accuracy and classification accuracy.

6.1.1 Prediction Accuracy

Prediction accuracy is used to evaluate how close the predicted ratings are to the observed ratings. Typical metric of this type is the mean absolute error(MAE) and root mean squared error(RMSE). MAE can be used to measure the average absolute deviation between the predicted ratings and the observed ratings. RMSE is another prediction accuracy metric that is similar to MAE. One major difference is that RMSE focuses more on large errors. We will use both of them in our evaluation on rating prediction. Formally,

$$\mathrm{MAE} = \frac{\sum_{(u,i)\in I} |r_{ui} - r'_{ui}|}{N}$$

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{(u,i)\in I} (r_{ui} - r'_{ui})^2}{N}}$$

The lower the MAE and RMSE, the more accurately the recommendation engine predicts ratings.

6.1.2 Classification Accuracy

Classification accuracy is used to measure how well the system makes decisions about good items and bad items. The most popular metrics include precision, recall, F-measure and coverage. Generally, they are appropriate for finding good items task. In order to apply them in the Top-N recommendation task, good or bad items are defined as the item that is rated or the item that is not rated respectively.

For simplicity, Table 2 illustrates the definition and relationships of these accuracy metrics. Recall focuses on how many of the positives does the model return; precision measures how many of the returned documents are correct. F-measure is the harmonic mean of precision and recall. Coverage measures the proportion of the total number of distinct items recommended across all users. Coverage reflects the ability of the recommendation algorithm in mining long tail items. The higher the coverage, the higher probabilities that the system will recommend long tail items to users. This project uses the precision, recall and coverage to measure the performance of our approach.

		True Rating	
		Good	Bad
Predicted Rating	Good	True Positive	False Positive
	Bad	False Negative	True Negative

Table 2:Illustration of Classification Accuracy

$$\text{precision} = \frac{\text{True Positive}}{\text{True Positive} \ + \ \text{False Positive}}$$

$$\text{recall} = \frac{\text{True Positive}}{\text{True Positive} \ + \ \text{False Negative}}$$

$$\text{F} - \text{measure} = 2 * \frac{\text{presision} * \text{recall}}{\text{presision} \ + \ \text{recall}}$$

$$\text{coverage} = \frac{\text{True Positive} \ + \ \text{False Positive}}{\text{The Total Number of Samples}}$$

6.2 Varying the Configuration Parameters

As described earlier, the accuracy of the latent factor models strongly depends on the dimension of hidden features f and learning rate γ, too little f or γ might not have enough explanatory power to differentiate the baseline algorithms and our approach. In the contrary, relatively large f or γ might lead to over-fitting.

In order to explore the effectiveness of the baseline algorithms and find the optimal parameters, we run various tests to analyze how the RMSE is correlated with the three parameters: f, γ and λ. First, we set the default values to be 10, 0.035, and 0.05 respectively, which are the values that produced more efficient and accurate results based on the preliminary trial runs, then we control two of the three parameters while varying one to explore its effect on RMSE. We plot RMSE of both the training dataset and the test dataset against $f \in [0, 1, 2, ..., 35]$, $\gamma \in [0.001, 0.002, 0.003, ..., 0.1]$, and $\lambda \in [0.01, 0.02, 0.03, ..., 0.5]$. Fig. 7 indicates that the RMSE is generally negatively correlated with f. By stepwise increasing the learning rate, RMSE gradually falls down until reaching the global minimum, and then continuously climbs up again. Fig. 7 also shows there is an obvious trend that RMSE increases with λ after reaching the global minimum. Based on the experimental results, the following configuration parameters are determined:

$$f = 25 \quad \gamma = 0.04 \quad \lambda = 0.1$$

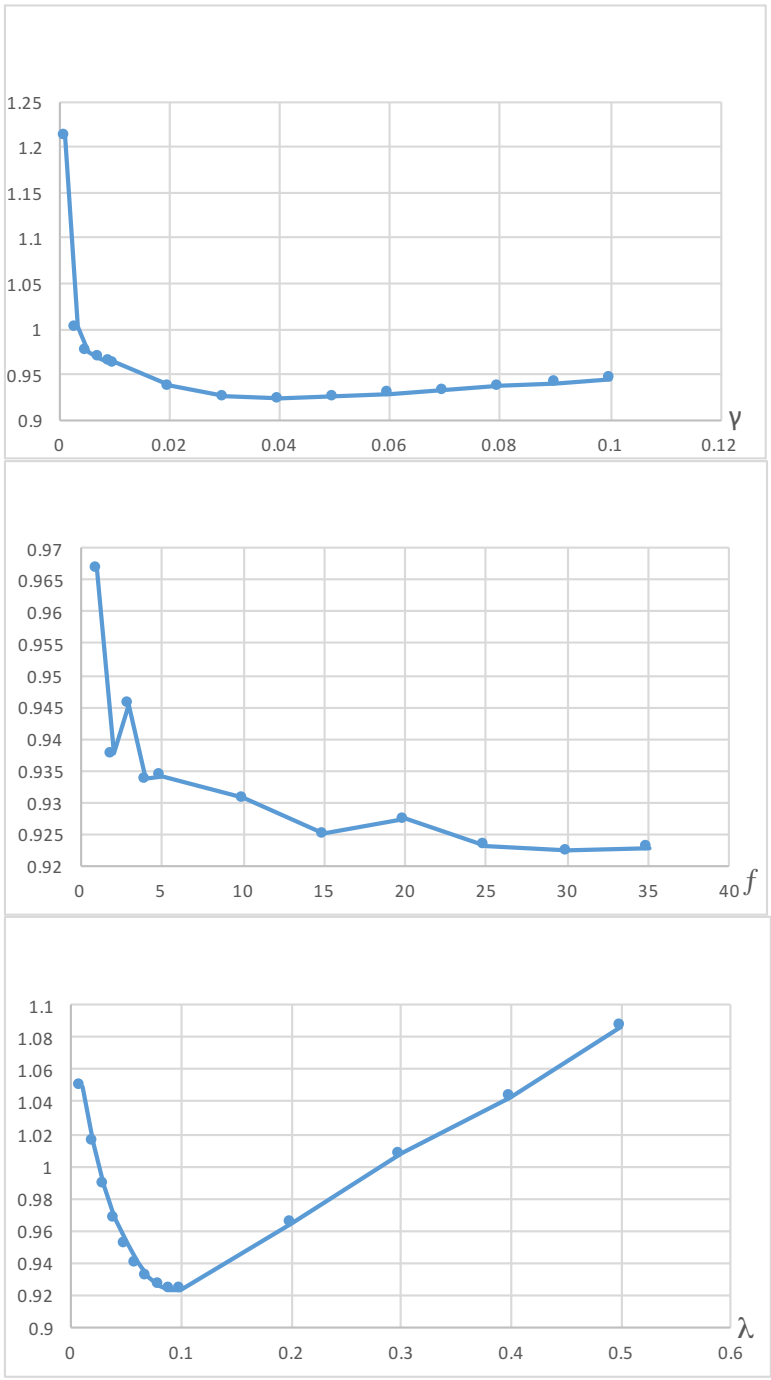

Figure 7:Relationship between RMSE and configuration parameters

6.3 Rating Prediction

In this part, we intend to validate whether our approach outperforms the other two baseline algorithms, Matrix Factorization and BSVD, in rating prediction. The evaluation metrics we use

are RMSE and MAE. The test results are summarized in Fig.8.

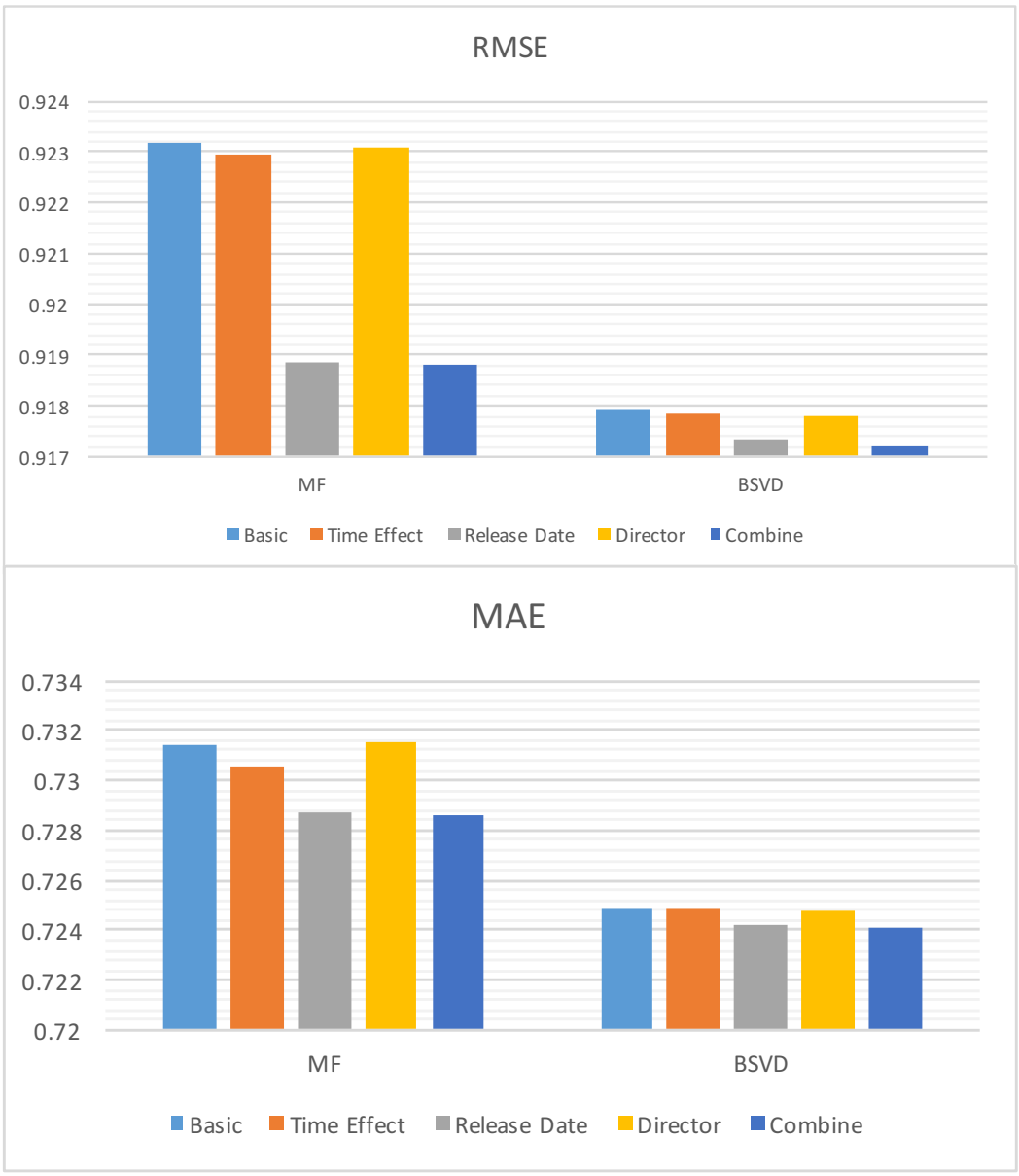

Figure 8: Performance comparison between our approach and the baseline algorithms. The results show that our approach is more efficient in rating prediction.

As can be seen from the results all of the Impact Factors bring improvements. We notice that in general release date factor significantly improves the prediction quality in both metrics, which justifies our previous conjecture that movie elements play an important role in user's rating decision. On the contrary, the improvement by integrating time effect with the models is not obvious, it might be because the time span of the 100K datasets is not big enough to reveal the

impact of popularity. The director factor is beneficial more to BSVD than to MF, the best prediction results can be reached by the combination of release date factor and director factor.

6.4 Top-N recommendation

Top-N recommendation focuses on the issue whether user will select the item or not. It is formally defined as follows [19]:

> Definition 1 (Top-N Recommendation Problem). Given a user–item matrix R and a set of items U that have been purchased by a user, identify an ordered set of items X such that $|X| \leq N$ and $X \cap U = \varnothing$.

Latent factor models cannot be applied to Top-N recommendation directly because the observed rating behavior only contains positive samples. In order to apply LFM to Top-N recommendation, we need to generate some negative samples for each user first. Pan et al. [20] proposed different methods to generate negative samples such as user-oriented sampling, item-oriented sampling, etc. In this project, we will randomly choose a certain amount of samples from the unknown rating dataset R_0 as the negative samples. Eq. (9) can also be applied to predict user's binary behavior. In such case, r'_{ui} denotes the prediction of the probability user u may rate item i, $r_{ui}= 1$ if the user-item pair (u, i) belongs to R and $r_{ui}= 0$ if the user-item pair (u, i) belongs to R_0. The algorithm for Top-N recommendation is straightforward. First, train the recommendation parameters with Algorithm 1, then compute the probability for every user-item pair in R_0. Finally, recommend the Top-N items with the highest probabilities for every user.

In the experimental processes, we set N=10 and the ratio of positive samples to negative samples is 6:1, then randomly select the negative samples from R_0 according to the ratio. To evaluate our work, we randomly sample (without replacement) 10 observed rating for each user from training datasets, and take them as testing data. The evaluation metrics are computed by comparing the Top-10 recommended movies against the movies in the testing dataset. The experimental results are shown in Fig. 9. It is important to note that since the number of movies in the recommendation list is equal to the number of the positive samples in our evaluation, recall and precision will have the same value.

The improvement made by the integration of item attributes is a little disappointing. Only the director factor brings slight improvement in precision. Although the release date factor gets noticeably higher score in rating prediction, it introduces noises in Top-N recommendation. The reason for this is still uncertain, but it reveals that predicted ratings by itself is not a perfect solution for Top-N recommendation problem. In addition, it might lead to the recommendation list too niche or unfamiliar to users, and exclude items that the users really interested in even though they

may not rate them highly. The ranking prediction model built by Netflix uses both predicted rating and popularity in order to compensate for the unbalance.

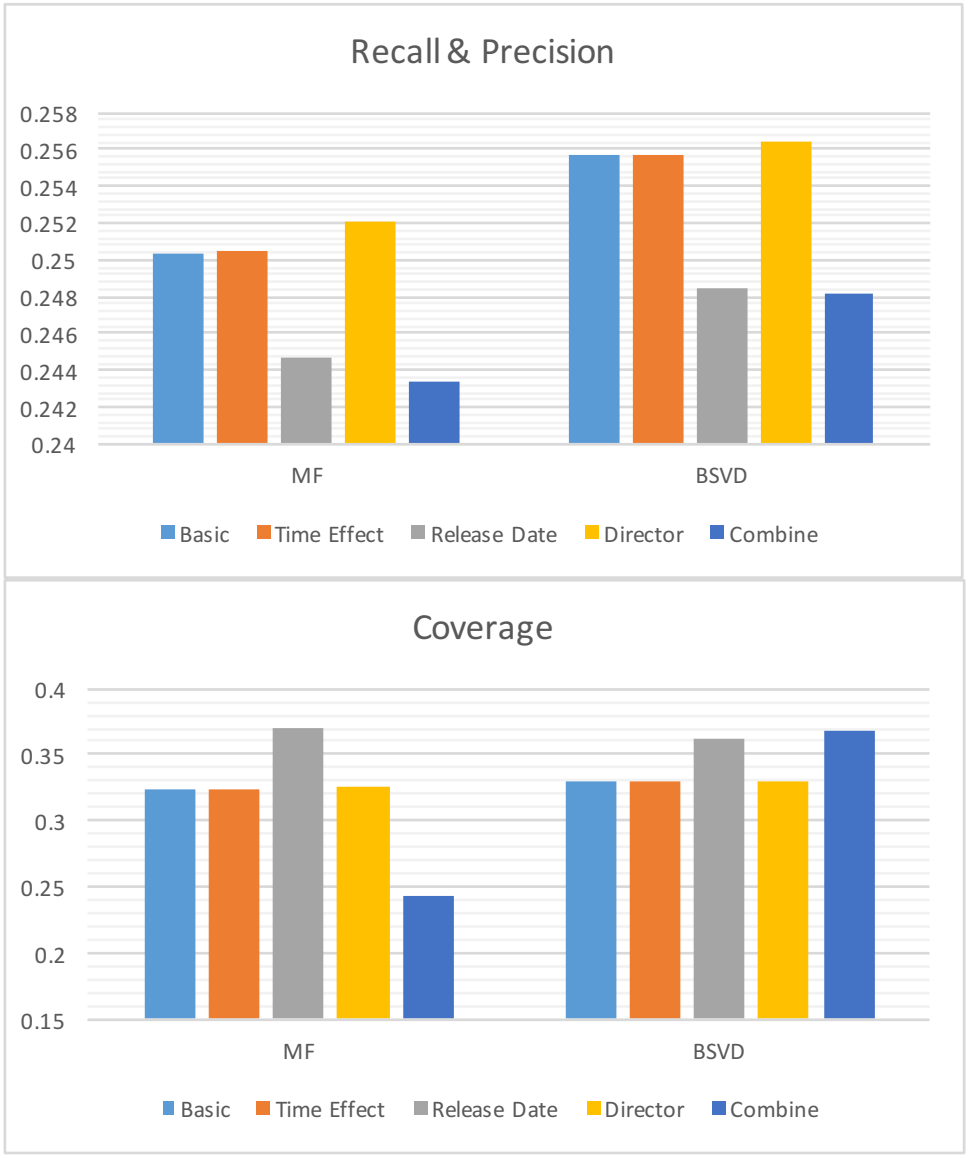

Figure 9: Performance comparison between our approach and the baseline algorithms in Top-N recommendation

7 Conclusions and future work

In this project we propose a lightweight approach to improving the recommendation accuracy, which combines Latent Factor Models with item attributes that influence user's taste preference on item. The main advantages of this approach are the maximum utilization of existing data and high extensibility. The approach is implemented in the Python programming language and tested by means of the well-known MovieLens rating dataset. The experimental results show that the Impact Factors are beneficial to the rating prediction for the movie recommender systems.

Predicting which items users will rate is more important than predicting which rating the active user will assign to the given items. Top-N recommendation is practical problems for many on-line recommender systems. The item attributes in our model have little impact on final results. In future, we will try to add other features such as the number of rating on the item, mean and variance ratings on the item to improve the recommendation accuracy. In addition, time ordering of user's rating data is also useful in Top-N recommendation problem as user's taste and mood may change. So if two items are close in the rating sequence, they may have large similarity. We will verify the conjecture in our future work.

Taking user information into consideration is another direction for improvement in future work. Users in recommender systems are usually associated with annotated attributes such as address, age, interests, etc. These attributes are highly informative and can be exploited for accurate recommendation. In future, we will explore an approach to learning user preference model over these attributes for implementing interpretable recommendation.

8 References

[1] Koren, Y., & Bell, R. (2011). Advances in collaborative filtering. In Recommender Systems Handbook (pp. 145-186). Springer US.

[2] Bell, R. M., & Koren, Y. (2007). Lessons from the Netflix prize challenge. ACM SIGKDD Explorations Newsletter, 9(2), 75-79.

[3] Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. Computer, (8), 30-37.

[4] Huang, Z., Zeng, D., & Chen, H. (2007). A comparison of collaborative-filtering recommendation algorithms for e-commerce. IEEE Intelligent Systems, (5), 68-78.

[5] Lee, S. K., Cho, Y. H., & Kim, S. H. (2010). Collaborative filtering with ordinal scale-based implicit ratings for mobile music recommendations. Information Sciences, 180(11), 2142-2155.

[6] Zhou, Y., Wilkinson, D., Schreiber, R., & Pan, R. (2008). Large-scale parallel collaborative filtering for the Netflix prize. In Algorithmic Aspects in Information and Management (pp. 337-348). Springer Berlin Heidelberg.

[7] Ricci, F., Rokach, L., & Shapira, B. (2011). Introduction to recommender systems handbook (pp. 1-35). Springer US.

[8] Grossman, L. (2010). How Computers Know What We Want—Before We Do. Time Magazine, 27.

[9] Byers, J. W., Mitzenmacher, M., & Zervas, G. (2010, February). Adaptive weighing designs for keyword value computation. In Proceedings of the Third ACM International Conference on Web Search and Data Mining (pp. 331-340). ACM.

[10] Pazzani, M. J., & Billsus, D. (2007). Content-based recommendation systems. In the adaptive web (pp. 325-341). Springer Berlin Heidelberg.

[11] Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2000). Application of dimensionality reduction in recommender system-a case study (No. TR-00-043). Minnesota Univ Minneapolis Dept of Computer Science.

[12] Wright, J., Ganesh, A., Rao, S., Peng, Y., & Ma, Y. (2009). Robust principal component analysis: Exact recovery of corrupted low-rank matrices via convex optimization. In advances in neural information processing systems (pp. 2080-2088).

[13] Zheng, Z., Chen, T., Liu, N. N., Yang, Q., & Yu, Y. (2012). Rating Prediction with Informative Ensemble of Multi-Resolution Dynamic Models. In KDD Cup (pp. 75-97).

[14] Zhou, Y., Wilkinson, D., Schreiber, R., & Pan, R. (2008). Large-scale parallel collaborative filtering for the Netflix prize. In Algorithmic Aspects in Information and Management (pp. 337-348). Springer Berlin Heidelberg.

[15] Ekstrand, M. D., Ludwig, M., Konstan, J. A., & Riedl, J. T. (2011, October). Rethinking the recommender research ecosystem: reproducibility, openness, and lenskit. In Proceedings of the Fifth ACM Conference on Recommender Systems (pp. 133-140). ACM.

[16] Lakshminarayanan, B., Bouchard, G., & Archambeau, C. (2011). Robust Bayesian matrix factorization. In International Conference on Artificial Intelligence and Statistics (pp. 425-433).

[17] Yu, X., Ren, X., Sun, Y., Sturt, B., Khandelwal, U., Gu, Q., ... & Han, J. (2013, October). Recommendation in heterogeneous information networks with implicit user feedback. In Proceedings of the 7th ACM Conference on Recommender Systems (pp. 347-350). ACM.

[18] Massa, P., & Avesani, P. (2004). Trust-aware collaborative filtering for recommender systems. In On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE (pp. 492-508). Springer Berlin Heidelberg.

[19] Deshpande, M., & Karypis, G. (2004). Item-based top-n recommendation algorithms. ACM Transactions on Information Systems (TOIS), 22(1), 143-177.

[20] Pan, R., Zhou, Y., Cao, B., Liu, N. N., Lukose, R., Scholz, M., & Yang, Q. (2008, December). One-class collaborative filtering. In Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on (pp. 502-511). IEEE.

[21] Amatriain, X., & Basilico, J. (2012). Netflix recommendations: beyond the 5 stars (part 1). Netflix Tech Blog, 6.

[22] Ahmed, A., Kanagal, B., Pandey, S., Josifovski, V., Pueyo, L. G., & Yuan, J. (2013, February). Latent factor models with additive and hierarchically-smoothed user preferences. In Proceedings of the Sixth ACM International Conference on Web Search and Data Mining (pp. 385-394). ACM.

[23] Karatzoglou, A., Amatriain, X., Baltrunas, L., & Oliver, N. (2010, September). Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In Proceedings of the Fourth ACM Conference on Recommender Systems (pp. 79-86). ACM.

[24] Xiong, L., Chen, X., Huang, T. K., Schneider, J. G., & Carbonell, J. G. (2010, January). Temporal Collaborative Filtering with Bayesian Probabilistic Tensor Factorization. In SDM (Vol. 10, pp. 211-222).

[25] Koren, Y. (2010). Collaborative filtering with temporal dynamics. Communications of the ACM, 53(4), 89-97.

[26] Bennett, J., & Lanning, S. (2007, August). The Netflix prize. In Proceedings of KDD Cup and Workshop (Vol. 2007, p. 35). Chicago